НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

Факультет інформатики та обчислювальної техніки

Кафедра обчислювальної техніки

До захисту допущено:
Завідувач кафедри
_____ Сергій СТІРЕНКО
«___»_____20__ р.

Дипломний проєкт

на здобуття ступеня бакалавра
за освітньо-професійною програмою «Комп'ютерні системи та мережі»
спеціальності 123 «Комп'ютерна інженерія»
на тему: «Комп'ютерна гра в жанрі МОВА»

Виконав:
студент IV курсу, групи ІО-62
Бурбіль Максим Андрійович _____

Керівник:
Доцент, кандидат технічних наук,
Верба Олександр Андрійович _____

Консультант з нормоконтролю:
Професор, доктор технічних наук
Сімоненко Валерій Павлович _____

Рецензент:
Доцент, кандидат технічних наук
Орлова Марія Миколаївна _____

Засвідчую, що у цьому дипломному проєкті немає запозичень з праць інших авторів без відповідних посилань.
Студент _____

Київ – 2020 року

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
Факультет інформатики та обчислювальної техніки
Кафедра обчислювальної техніки

Рівень вищої освіти – перший (бакалаврський)

Спеціальність – 123 «Комп'ютерна інженерія»

Освітньо-професійна програма «Комп'ютерні системи та мережі»

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Сергій СТІРЕНКО
«___»_____20__ р.

ЗАВДАННЯ
на дипломний проєкт студенту
Бурбілю Максиму Андрійовичу

1. Тема проєкту «Комп'ютерна гра в жанрі МОВА», керівник проєкту Верба Олександр Андрійович, доцент, к.т.н., затверджені наказом по університету від «07» травня 2020 р. № 1081-с

2. Термін подання студентом проєкту 06 червня 2020 р.

3. Вихідні дані до проєкту: технічне завдання, науково-технічна література

4. Зміст пояснювальної записки: порівняльний аналіз існуючих програмних рішень, вибір засобів реалізації та опис отриманої системи

5. Перелік графічного матеріалу (із зазначенням обов'язкових креслеників, плакатів, презентацій тощо) :

1. Функціональна схема – плакат
2. Принципова схема – плакат
3. Структурна схема – плакат

6. Консультанти розділів проєкту

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Нормоконтроль	Сімоненко В.П., професор		

7. Дата видачі завдання 01 вересня 2019 р.

Календарний план

№ з/п	Назва етапів виконання дипломного проєкту	Термін виконання етапів проєкту	Примітка
1	Затвердження теми роботи	01.09.2019-22.12.2019	
2	Вивчення та аналіз завдання	23.12.2019-20.03.2020	
3	Розробка архітектури та загальної структури системи	20.03.2020-01.04.2020	
4	Розробка структур окремих підсистем	01.04.2020-10.04.2020	
5	Програмна реалізація системи	11.04.2020-20.04.2020	
6	Оформлення пояснювальної записки	01.05.2020-23.05.2020	
7	Передзахист	24.05.2020-26.05.2020	
8	Захист	15.06.2020-20.06.2020	

Студент Максим БУРБІЛЬ

Керівник Олександр ВЕРБА

АНОТАЦІЯ

Дана дипломна робота присвячена розробці комп’ютерної гри в жанрі MOBA на базі існуючого ігрового рушія, що призначена для проведення дозвілля.

Гравець може обирати собі героя та змагатись з іншими героями, що контролюються іншими гравцями або ж штучним інтелектом, на спеціальній мапі, що має назву - арена.

Для реалізації гри використовується ігровий рушій Unreal Engine 4 та мова програмування C++.

ANNOTATION

This thesis is devoted to the development of a computer game in the genre of MOBA on the basis of the existing game engine, which is designed for leisure.

The player can choose a hero and compete with other heroes controlled by other players or artificial intelligence on a special map called the arena.

To implement the game, the game engine Unreal Engine 4 and the C ++ programming language are used.

ВІДОМІСТЬ ДИПЛОМНОГО ПРОЄКТУ

№ з/п	Формат	Позначення	Найменування	Кількість листів	Примітка
1	А4		Завдання на дипломний проєкт	2	
2	А4	*ІАЛЦ. 467800.001* ВП	Відомість проєкту	1	
3	А4	*ІАЛЦ. 467800.002* ТЗ	Технічне завдання	3	
4	А4	*ІАЛЦ. 467800.003* ПЗ	Пояснювальна записка	56	
5	А4	*ІАЛЦ. 467800.004* Д1	Функціональна схема	1	
6	А4	*ІАЛЦ. 467800.005* Д2	Принципова схема	1	
7	А4	*ІАЛЦ. 467800.006* Д3	Структурна схема	1	
8	А4	*ІАЛЦ. 467800.007* Д4	Текст програми	20	

				ІАЛЦ. 467800.001 ВП	
	ПІБ	Підп.	Дата		
Розробн.	Бурбіль М.А.			Відомість дипломного проєкту	Лист 1 / Листів 1
Керівн.	Верба О.А.				
Консульт.					КПІ ім. Ігоря Сікорського Каф. ОТ Гр. ІО-62
Н/контр.	Сімоненко В.П.				
Зав.каф.	Стіренко С.Г.				

Технічне завдання

до дипломного проєкту

на тему: «Комп’ютерна гра в жанрі МОВА»

Київ – 2020 року

ЗМІСТ

1. НАЙМЕНУВАННЯ ТА ОБЛАСТЬ ЗАСТОСУВАННЯ

Дане технічне завдання розповсюджується на розробку програмного додатку на тему «Комп’ютерна гра в жанрі МОВА».

Область застосування: організація дозвілля та розвиток різноманітних навичок гравця.

2. ПІДСТАВИ ДЛЯ РОЗРОБКИ

Підставою для розробки служить завдання на виконання роботи кваліфікаційно-освітнього рівня «бакалавр комп'ютерної інженерії», затверджене кафедрою обчислювальної техніки Національного технічного Університету України «Київський Політехнічний інститут ім. Ігоря Сікорського».

3. МЕТА ТА ПРИЗНАЧЕННЯ РОЗРОБКИ

Метою даного проекту є розробка ігрового додатку, що слугує для організації дозвілля та розвитку різноманітних навичок.

4. ДЖЕРЕЛА РОЗРОБКИ

Джерелами розробки є технічна документація для інтерфейсів прикладного програмування та бібліотек, що використовуються під час розробки програмного продукту, науково-технічна література з інформаційних технологій, публікації в періодичних виданнях, а також відповідні статті в мережі Інтернет за даним питанням.

5. ТЕХНІЧНІ ВИМОГИ

5.1. Вимоги до програмного продукту, що розробляється

- Зберігання даних на пристрої.
- Конкурентоспроможність ігрового додатку на ринку комп’ютерних ігор.
- Зручність і простота графічного інтерфейсу розроблюваної системи.

5.2. Вимоги до програмного забезпечення

- Мова програмування C++.
- Використання ігрового рушія Unreal Engine 4.

5.3. Вимоги до апаратного забезпечення

- Комп’ютер на базі процесору Intel Pentium 4 / Athlon 64 і вище.
- Оперативної пам’яті не менше 4096 Мбайт.
- 3.5 Гбайт вільного місця на пристрої зберігання інформації.
- Підключення до мережі Інтернет.

Пояснювальна записка

до дипломного проєкту

на тему: «Комп’ютерна гра в жанрі МОВА»

Київ – 2020 року

ЗМІСТ

ПЕРЕЛІК ТЕРМІНІВ ТА СКОРОЧЕНЬ

ОС	Операційна система
ПК	Персональний комп’ютер
GE	(англ. Game Engine) центральна програмна частина будь-якої комп’ютерної гри, яка відповідає за всю технічну сторону
MOBA	(англ. Multiplayer Online Battle Arena, Багатокористувацька онлайнова бойова арена) – жанр комп’ютерної гри
RPG	(англ. Role-Playing Game, Рольова відеогра) – жанр відеоігор
RTS	(англ. real-time strategy, Стратегія в реальному часі) – жанр комп’ютерних ігор
fps	(англ. frames per second, частота кадрів – швидкість з якою пристрій, що формує зображення відображає послідовні зображення, що мають назву кадри)
IDE	(англ. Integrated development environment, Інтегроване середовище розробки) — комплексне програмне рішення для розробки програмного забезпечення
SDK	(англ. Software Development Kit) — набір із засобів розробки, утиліт і документації, за допомогою якого програмісти можуть створювати програми за визначеною технологією

RPC	(англ. Remote Procedure Call, Виклик віддалених процедур) – технологія, що дозволяє програмі на одному комп’ютері запускати функції (процедури) програми на іншому комп’ютері.
ЧІВ	часовий інтервал, протягом якого вміння персонажу не можна застосовувати, вводиться для підтримки ігрового балансу та поглиблення ігрових механік.

ВСТУП

З появою персональних комп'ютерів, з кожним роком, їх роль в житті людей постійно зростає. Комп'ютер став незамінним помічником не тільки в науковій та економічній сферах, а й є потужним центром розваг. Сфери впливу кіно і літератури відчувають потужний тиск з боку інтерактивних розваг, пристроїв доповненої реальності, ігрових консолей та багатьох інших, впроваджуваних завдяки розвитку комп'ютерних технологій.

Комп'ютерні розваги роблять життя людини багатше, більш насиченим і як наслідок - це потужна економічна сфера, що приносить прибуток.

Тому не випадково, що особлива роль у житті сучасної людини відводиться комп'ютерним іграм, перші з яких з’явились ще на початку історії комп’ютерної техніки.

Можна сказати, що ігри - це певний вид мистецтва, схожий з іншими жанрами. Ігри можуть нести не тільки розважальний характер, але і змушувати задуматися, переживати, піднімати серйозні глобальні або психологічні питання. Іншими словами, ігри - це сучасний вид мистецтва. Комп'ютерні ігри часом дарують емоцій не менше, ніж перегляд кінофільму або театральної вистави чи відвідування концерту.

Актуальність теми

У світі існує надзвичайно велика кількість поціновувачів комп'ютерних ігор з різним досвідом в цій сфері. Є люди, які тільки но відкрили для себе всю різнобарвність комп’ютерних ігор і їх жанрів, а, також, є навчені досвідом гравці, які уже обрали для себе улюблені жанри, тому створення комп’ютерних ігор буде завжди затребуваним на цьому ринку.

Мета і задачі дослідження

Метою даної дипломної роботи є створення комп’ютерної гри в жанрі МОВА, використовуючи готовий GE, що дозволяє створювати ігри на високому рівні програмування, не задумуючись про взаємодію з апаратним забезпеченням.

Для досягнення поставленої мети були поставлені наступні основні задачі:

- Провести аналіз існуючих ігор в жанрі МОВА, відмітити їх позитивні та негативні сторони.
- Провести аналіз існуючих GE та обрати найбільш оптимальний для розробки даної гри.
- Створити модель та програмну реалізацію комп’ютерної гри використовуючи можливості обраного GE;
- Дослідити та проаналізувати отримані характеристики під час тестування.

РОЗДІЛ 1.
ПРЕДСТАВЛЕННЯ ГОТОВИХ РІШЕНЬ

1.1 Загальні відомості про комп'ютерні ігри

Перед тим як переходити до суті мого дипломного проекту, необхідно зрозуміти, що являє собою комп'ютерна гра. І так – це програма, створена для організації ігрового процесу (геймплею), зв'язку з партнерами по грі (іншими гравцями). В наш час для визначення комп'ютерної гри можна використовувати термін «відеогра», тобто він слугує її синонімом. Пристрої, що використовуються для гри називаються ігровими платформами, такими можуть бути ПК або ігрова консоль. Для керування відеогрою гравець повинен використовувати пристрої введення, які в даній галузі носять назву «Ігрові контролери» (наприклад, клавіатура та мишка, джойстик, геймпад та багато інших). На даний момент комп'ютерні ігри можуть класифікувати за жанрами, до яких можна віднести геймплей гри. Прикладом жанрів можуть слугувати RTS (основна геймплею полягає в відсутності почергових ходів, тобто гравець і супротивник виконують дії одночасно), RPG (основна частина геймплею полягає в управлінні одним або декількома персонажами, що досліджують ігровий світ, виконують завдання та розвиваються), MOBA. Також, дані ігри можуть бути використані не лише в розважальних цілях, а й в навчальних програмах, та навіть в спорті. З появою комп'ютерних ігор створилась нова галузь спортивних змагань – кіберспорт.

Ігри можна віднести до категорій за кількість гравців. Існують одиночні ігри (single-player, гравець грає сам, без інших живих гравців, може бути присутній штучний інтелект), кооперативні (зазвичай підтримують 2-3 гравців, в яких є поставлена ціль, що досягається спільними зусиллями або ж змагання між гравцями), багатокористувацькі (де грає більше одного гравця, кооперативні відносяться як піджанр багатокористувацьких ігор). Багатокористувацькі ігри (також відомі як Мультиплеєрні ігри) можуть бути розділені на підкатегорії за технічною реалізацією:

- Гравці можуть грати на одному комп’ютері. Гравці можуть одночасно управляти, при такому розкладі екран розділяється на декілька частин, ця технологія має назву splitscreen (застосовується в спортивних симуляторах). Друга можливість коли гравці ходять по черзі на одному й тому самому комп’ютері. Така технологія називається hotseat і використовується в покрокових стратегіях.
- Онлайн ігри. Гравці грають на окремих комп’ютерах через мережу Інтернет через протоколи IPX або TCP/IP (більшість сучасних ігор).

1.2 Приклади відеоігор в жанрі MOBA

Перед порівнянням ігор я вважаю за потрібне сказати декілька слів про сам жанр MOBA (англ. *Multiplayer Online Battle Arena,* Багатокористувацька онлайнова бойова арена) - це жанр комп'ютерних ігор, який поєднує в собі елементи таких жанрів, як RTS та RPG.

В іграх жанру MOBA йде протистояння двох команд. Кожен гравець може управляти лише одним або декількома героями під час одного матчу, якого він зазвичай обирає перед початком бою, при чому кожен з героїв має свої унікальні характеристики та здібності. Протягом матчу герой, під керівництвом гравця, може вивчати нові навички та покращувати свої характеристики. Незважаючи на описане вище, межі даного жанру є доволі розмитими, оскільки сама назва жанру може трактуватися кожним розробником по своєму. Саме тому кінцева мета в таких іграх може відрізнятися геймплею самої гри. Та зазвичай – це знищення головної будівлі однієї з команд. До цього жанру входять такі ігри, як Dota 2, League of Legends, Battlerite, Prime World.

Оскільки кінцева мета може бути різною, я обрав для порівняння дві гри з цього жанру, які кардинально відрізняються одна від одної, щоб показати всю різнобарвність MOBA - жанру.

1.2.1 Dota 2

Рисунок 1.1 – Приклад геймплею Dota 2

Dota 2 – комп’ютерна гра, створена компанією Valve в 2013 році, що використовувала власний GE - Source 2, одна з кіберспортивних дисциплін на даний момент. Дана гра являється, яскравим представником цього жанру, оскільки в ній працюють всі правила, що були описані вище. Команди складаються з 5 гравців, кожен з яких, контролює одного героя з різними можливостями та характеристиками, якого він може обрати з обширного списку героїв перед початком матчу. Матч в Dota 2 закінчується тоді, коли фортеця, особливий об’єкт однієї з команд буде зруйнованою. Сам геймплей гри заточений під постійну взаємодію з командою, та розрахунку економічної ситуації своєї команди, оскільки спорядження для героїв купується на ігрову валюту - золото, що здобувається при знищенні ворожих героїв, істот – «крипів» та їх будівель. Кожен матч проходить на квадратній мапі спеціального виду, де фортеці обох команд знаходяться в протилежних кутках, а гравці розосереджуються по шляхам, що з’єднують ці фортеці - «лініях». [6]

Рисунок 1.2 – Вигляд мапи в Dota 2

Крім самих гравців, в грі беруть участь керовані комп'ютером істоти - «крипи» і нерухомі будівлі - «вежі», свої з кожного боку; вони також беруть участь в битві, атакуючи ворожих героїв, крипів та вежі противника, тим самим допомагаючи своїй команді. Туман війни, що покриває велику частину карти, не дозволяє гравцям стежити за пересуваннями противника.[6]

Dota 2 являється класичним представником жанру MOBA і не відступає від основної класифікації.

1.2.2 Battlerite

Рисунок 1.3 – Приклад геймплею Battlerite

Battlerite – комп’ютерна гра, створена в 2016 році компанією Stunlock Studios. Для створення цієї відеогри були використані технології ігрового рушія Unity. Дана гра відрізняється своїм геймплеєм від основної класифікації MOBA, залишається лише незмінною система прив’язки гравця до одного героя. В грі присутні дві команди, в яких може бути від 2 до 3 гравців, залежно від обраного режиму. Основною метою в Battlerite являється перемога вашої команди, шляхом знищення ворожої команди на протязі 5 раундів. Даний режим має свою назву BO5 (англ. Best of 5), перемагає команда яка набрала 3 перемоги. Після кожного раунду йде відродження героїв на їх початкові бази і раунди починаються з початку. Перед першим раундом гравець може обрати додаткові бойові здібності, що впливають на хід гри і можуть бути заточені на протидію конкретному герою опонента. Всі раунди відрізняються між собою лише стратегією, яку гравці виберуть для своєї команди.[7]

ВИСНОВОК ДО РОЗДІЛУ 1

В ході виконання даного розділу моєї дипломної роботи, було розглянуто декілька ігор в жанрі МОВА. Щодо плюсів та мінусів ігор, які я описав, то не можливо виділити конкретні досягнення чи проблеми, оскільки ідею та геймплейні механіки створює розробник власноруч, притримуючись деяких границь, щоб віднести свою гру до певного жанру. Саме тому, я вкажу те, що особисто мені не подобається в геймплеї кожної з цих ігор і щоб я хотів виправити або додати до цих механік.

Для Dota 2 я вважаю, геймплей занадто затягнутим, оскільки більшу частину гри виділяється на підвищення рівня героя та спорядження, а також на загально економічній ситуації в команді. Як результат, один матч може тривати від 30 хвилин і більше. Найбільш тривалий матч в даній грі тривав 6 годин і 8 хвилин.

В свою чергу геймплей Battlerite є повною протилежністю Dota 2. Один матч замінений 5 раундами, кожен з яких в свою чергу має обмеження по часу і триває не більше 2 хвилин. В результаті даного рішення один матч триватиме не більше 10-15 хвилин. Та все ж, в даній грі система прив’язки гравця до окремого герою протягом матчу залишається незмінною, а, також, система BO5, яка на мою думку є чудовим рішенням для рейтингових ігор (команди змагаються між собою і займають певні місця в рейтингу, що складається з команд зі всього світу), але є не доцільною для звичайних матчів.

Саме тому я хочу розробити власну комп’ютерну гру в цьому жанрі, в якій не буде прив’язки до певного герою, а гравець буде мати можливість змінити його в матчі. Також, окрім можливості грати в команді, він зможе обрати режим «кожен сам за себе», де він зможе розраховувати лише на свої здібності. Зменшення тривалості матчу та зміни правил гри, повинно забезпечити більшу динаміку в геймплеї.

РОЗДІЛ 2
АНАЛІЗ ЗАСОБІВ ДЛЯ РОЗРОБКИ КОМП'ЮТЕРНОЇ ГРИ

2.1 Загальні відомості про ігрові рушії

На даний момент комп'ютерні ігри створюються на базі обраного розробником GE. Перед тим як переходити до порівняння різних GE слід зрозуміти, що вони представляють в собі.

Отже, GE (англ. *Game engine,* Ігровий рушій) - це програма, яка бере на себе керування всією технічною стороною будь-якої відеогри, дозволяє полегшити розробку гри шляхом уніфікації та систематизації її структури. На сьогоднішній день, одним з найважливіших значень даних рушіїв являється створення кросплатформних ігор (наприклад розробка гри одночасно для ПК, PS4 (Playstation 4 – ігрова приставка від компанії Sony) та Xbox One (ігрова приставка розроблена компанією Microsoft)). Увесь контент, що використовується програмістом під час розробки ігор (музика, графіка, анімації і т.д.) називаються асетами (англ. Assets). Основну функціональність гри зазвичай забезпечує її GE, який в свою чергу складається з декількох рушіїв (складних підсистем):

1) рушій рендерингу («візуалізатор») – основною задачею є представлення двовимірної або тривімірної комп'ютерної графіки;
2) фізичний рушій – підсистема, яка відповідає за симуляцію фізичних законів реального світу в віртуальному;
3) звук, система скриптів, анімація;
4) ігровий штучний інтелект – набір методик, які дозволяють створити ілюзію інтелекту в поведінці персонажів, що контролюються комп'ютером;
5) мережевий код – частина рушія, завдяки якій існує можливість створювати онлайн ігри. Часто це код, який відповідає за створення клієнту та серверу, та їх взаємодію між собою;

6) багатопотоковість;

7) граф сцени – така структура даних, що використовується в застосунках для роботи з векторною графікою;

2.2 Приклади сучасних GE, їх плюси та мінуси.

На даний момент в світі існує дуже багато різних GE. В цьому пункті я порівняю найбільш сучасні та відомі з них.

2.2.1 Unity

Unity – це не лише ігровий рушій, а й кросплатформне середовище розробки комп'ютерних ігор, що була розроблена американською компанією Unity Technologies. Дата першого релізу припала на 2005 рік, і з того часу цей GE невпинно розвивається. Технологія Unity дозволяє створювати різні додатки, що будуть працювати більш ніж на 25 платформах, найяскравішими з яких є ПК, ігрові консолі та мобільні телефони. Як було сказано, даний GE являється повноцінним середовищем для розробки комп'ютерних ігор, оскільки в ньому об'єднані різні програмні засоби, що використовуються при створенні ПО – текстовий редактор, компілятор, дебагер і т.д. Редактор Unity має простий та інтуїтивно зрозумілий інтерфейс, що підтримує технологію Drag & Drop завдяки якій його можна легко налаштовувати під користувача, а також проводити налагодження гри прямо в цьому редакторі.[1]

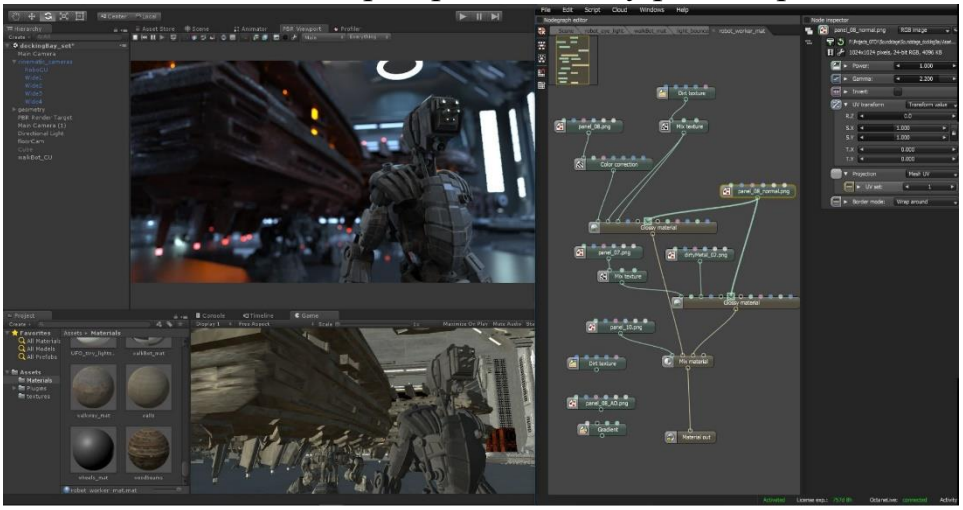


Рисунок 2.1 – Інтерфейс редактора Unity

Для написання скриптів в даному рушії використовується мова програмування C#, при цьому старіші версії цього GE підтримували модифікацію JavaScript (більш відома як UnityScript, підтримка була припинена в версії 2017.1), а також мову програмування Boo (діалект мови Python, підтримка була припинена в 5 версії ігрового рушія). На даний момент на Unity написані тисячі ігор, додатків, що охоплюють різні платформи та жанри і використовується він не лише незалежними розробниками (indie – розробниками), а й великими компаніями.[1]

Серед плюсів Unity можна відмітити наступне:

- В редакторі використовується компонентно-орієнтований підхід в рамках якого розробник створює об’єкт (наприклад, головного героя) і додає до нього різні компоненти (візуальне відображення персонажа, управління ним).
- Зручний Drag & Drop інтерфейс графічного редактора дозволяє малювати карти та розставляти об’єкти на них в режимі реального часу і відразу тестувати результат.
- Наявність великої бібліотеки асетів та плагінів, що дозволяють прискорити процес розробки гри.
- Підтримка великої кількості платформ, кроскомпіляція під різні ОС, такі як Windows, Linux, OS X, Android, IOS та під ігрові консолі PlayStation, Nintendo, Xbox, на пристрої віртуальної та доповненої реальності.

Що ж до мінусів, то вони наступні:

- Складність при роботі з багатокомпонентними схемами і ускладнення при підключенні сторонні бібліотеки.
- Повільність. Створення масштабних та складних сцен с багатьма компонентами може негативно вплинути на швидкодію гри, в результаті чого необхідно збільшити час на оптимізацію гри, можливо видалення деяких елементів з проекту.

- Додатки, створені на Unity займають багато місця на носіїї інформації. Навіть найпростіша гра може займати декілька сотень мегабайт на диску. І якщо для ПК це не дуже великий об’єм, то для мобільних платформ його слід оптимізувати.

Наостанок хотів сказати декілька слів про ліцензію даного ігрового рушія. Unity доступний безкоштовно, що дозволяє незалежним розробникам займатися розробкою ігор на готовому GE. Але існують в безкоштовній версії існують деякі обмеження, наприклад, перед запуском вашої гри буде демонструватись логотип Unity та дохід від гри повинен бути не більше 100 тис. доларів в рік. В іншому ж випадку, розробник зобов’язаний придбати професійну версію GE, що коштує 125 доларів за місяць або 1800 доларів.[1]

Приклади відеоігор, розроблених на цьому рушії: Rust, Ori and the Blind Forest, Subnautica.[23]

Рисунок 2.2 – Приклад гри на Unity. Rust [23]

Рисунок 2.3 – Приклад гри на Unity. Ori and the Blind Forest [23]

Рисунок 2.4 – Приклад гри на Unity. Subnautica [23]

2.2.2 Unreal Engine 4

Наступним в списку виступає Unreal Engine 4 (UE4) – ігровий рушій створений та підтримуваний компанією Epic Games. Цифра 4 означає поточну версію цього GE реліз якої відбувся 2014 році, найперша ж версія Unreal Engine 1 була випущена в 1998 році. Як і Unity, він підтримує можливість розробки ігор для більшості ОС та платформ. [2]

На відміну від Unity, UE4 підтримує розроблену Epic Games систему візуального програмування, що носить назву Blueprint являє собою повну

систему сценаріїв ігрових процесів, що базується на концепції використання інтерфейсу на основі блоків-вузлів для створення елементів ігрового процесу в редакторі Unreal Editor. Як і більшість скриптових мов, система використовується для визначення об'єктно-орієнтованих класів чи об'єктів в UE4, неформальною назвою яких стала Blueprints. [2]

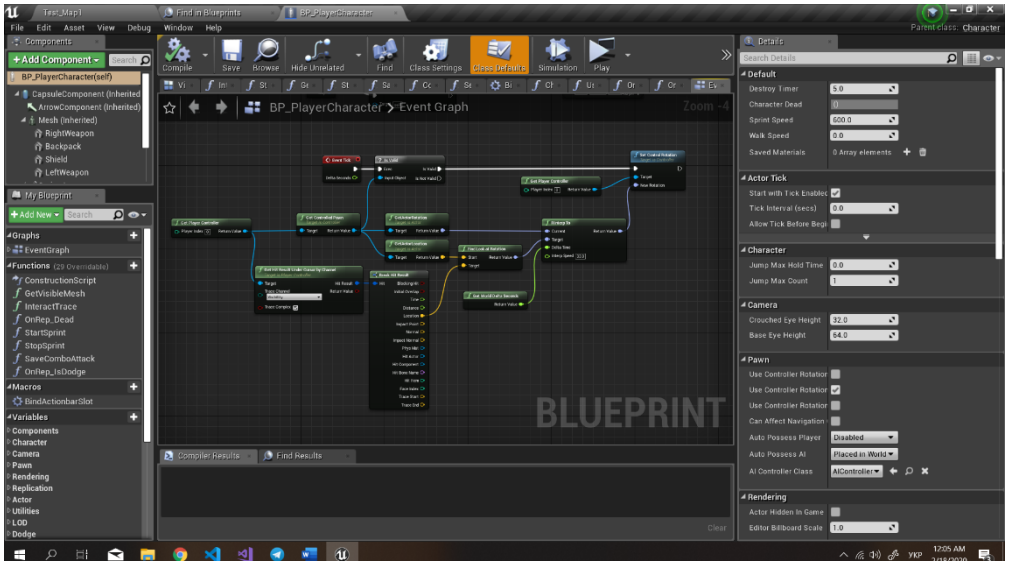


Рисунок 2.5 – Редактор Blueprint класів та загальний вигляд Blueprints

Ця система надзвичайно потужна і гнучка, оскільки дозволяє дизайнерам використовувати майже весь спектр концепцій та інструментів, які зазвичай доступні лише програмістам. Більш того, в UE4 існує спеціальна розмітка для Blueprint, що доступна в модифікації C++, підтримувана в Unreal Engine 4, яка дозволяє програмістам створювати базові системи, які будуть розширені дизайнерами. Також, існує можливість використовуючи C++ створювати власні блоки для своїх Blueprints. Деякі великі компанії використовують технології «Unreal» як основу для створення власного рушія. Так на основі Unreal Engine 2 був створений «Flesh Engine», на версії 2.5 створені Vengeance Engine компанією Irrational Games, LEAD engine та YETI engine створені компанією Ubisoft. [2]

З плюсів даного рушія можна відмітити наступні:

- Необов’язково потрібно мати знання C++ для того, щоб створити гру на UE4, оскільки система Blueprint дозволяє це зробити фактично не набираючи ні одного рядку коду.
- Система Blueprint дозволяє швидко створити прототип гри та протестувати його.
- Великий доступ до асетів та плагінів через платформу Unreal Marketplace. Для зацікавлення нових розробників Epic Games розпочала щомісячну роздачу платних асетів та плагінів безкоштовно.
- Розробники Unreal Engine 4 постійно спілкуються з своєю спільнотою, викладають навчальні матеріали, де пояснюють як працює той чи інший функціонал. Документація по цьому рушію є однією з найбільш обширних.

Серед мінусів можна вказати наступні:

- Проблеми з штучним інтелектом: при додаванні на рівень занадто багато істот з штучним інтелектом, спроби рушія обробити поведінку всіх одночасно призведе до падіння fps (англ. *frames per second*, частота кадрів – швидкість з якою пристрій, що формує зображення відображає послідовні зображення, що мають назву кадри), тому розробникам прийдеться винаходити способи обмеження діяльності істот з AI.
- Високий поріг входження для початківців в галузі розробки комп’ютерних ігор. Незважаючи на те, що гру можна створити не написавши ні одного рядку коду, все ж для людини у якої немає знань в даній галузі цей ігровий рушій може виявитись складним через обширний функціонал, що пропонує Unreal Engine 4 «з коробки».

Слід сказати пару слів про ліцензію даного рушія. З 2 березня 2015 року Unreal Engine 4 став безкоштовним. Але розробники повинні передавати 5%

від продаж гри компанії Epic Games, якщо квартальний дохід перевищує 3000 доларів. Також, сирцеві коди цього рушія доступні на веб-сервісі GitHub, і використовуються, наприклад для компіляції виділених серверів для ігор. [2]

Приклади комп'ютерних ігор розроблених, використовуючи технологію Unreal Engine: Mortal Kombat 11, Fortnite, Man of Medan.[21]

Рисунок 2.6 – Приклад гри на Unreal Engine. Mortal Kombat 11[21]

Рисунок 2.7 – Приклад гри на Unreal Engine. Fortnite [21]

Рисунок 2.8 – Приклад гри на Unreal Engine. Man of Medan [21]

2.2.3 CryEngine V

Останнім в моєму списку виступає ігровий рушій від німецької компанії Crytek - CryEngine V, що був анонсований в 2016 році. Він являється оновленою версією CryEngine 4, в якій було додана підтримка віртуальної реальності, та написання скриптів на C#. Розробка в даному GE відбувається на мові програмування C++, а також існує підтримка системи Flowgraph, що схожа на Blueprint від Epic Games. Та на відміну від останньої, Flowgraph набагато повільніший тому, що не генерує читаємий код і не може бути згенерований з коду. Через це не можливо створити складну систему, використовуючи дану технологію, оскільки вона буде не оптимізованою і з низьким fps. Існує можливість створювати власні блоки, але для цього необхідні знання C++ або C#. [3]

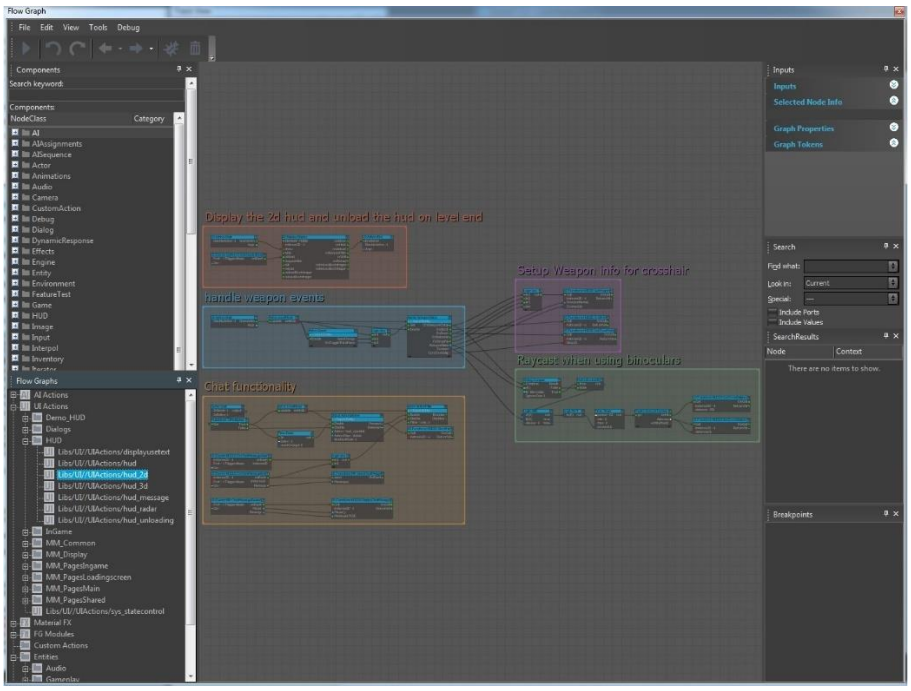


Рисунок 2.9 – Технологія Flowgraph в редакторі рушія CryEngine[3]

Ігровий рушій постачається безкоштовно разом з SDK, та з запуском версії 5.5 розробники повинні виплачувати 5% від прибутку, отриманих від ігор, що використовують останні версії даного GE, що перевищує 5000 доларів. Розробники, що використовують старіші версії рушія можуть подати заявку на звільнення від виплат, якщо в майбутньому вони не будуть обновлятися до версії 5.5 або вище. [3]

Серед плюсів CryEngine V можна виділити наступні:

- Підтримка передових технологій, таких як DirectX12, Vulkan API, VR, написання скриптів на мові програмування C#.
- Фотореалістична графіка, що при правильній розробці перевершує будь-які ігри створені за допомогою Unity або Unreal Engine 4. Також, рушій підтримує realtime render, що дозволяє швидко протестувати щойно створений рівень.
- GameSDK – інструмент, що постачається разом з ігровим рушієм, та на основі якого можливо швидко створювати власні ігри.
- Компанія Crytek розробила власну технологію рейтрейсингу (англ. *raytracing*, трасування променів – технологія, що дозволяє

створювати реалістичний спосіб рендерингу освітлення та тіней, шляхом симуляції і відслідковування кожного променя від джерела світла, і як результат вона вкрай вимоглива до апаратного забезпечення), яка працює на відеокартах компаній AMD та Nvidia і не потребує потужності графічних чипів RTX.

Що ж до мінусів, то вони наступні:

- Скромність вибору асетів та плагінів в CryEngine Marketplace в порівнянні з Unreal Marketplace або Unity Asset Store.
- Складність при компіляції фінальної версії продукту.
- Обмеження при розробці онлайн ігор, стандартна версія рушія підтримує не більше 32 гравців (ліміти, прописані в коді).
- Відсутність якісної технічної підтримки і мала за розмірами спільнота, в результаті чого багато проблем вирішуються методом спроб та помилок.

Декілька ігор створених за допомогою різних версій CryEngine: Crysis, Hunt: Showdown, Kingdom Come: Deliverance.[22]

Рисунок 2.10 – Приклад гри на CryEngine. Crysis [22]

Рисунок 2.11 – Приклад гри на CryEngine. Hunt: Showdown [22]

Рисунок 2.12 – Приклад гри на CryEngine. Kingdom Come: Deliverance [22]

2.3 Аналіз ігрового процесу комп’ютерних ігор в жанрі MOBA та модель власної реалізації

В першому розділі був зроблений короткий огляд ігор в жанрі MOBA та їх загальні характеристики та був запропонований власний варіант такої комп’ютерної гри. Тепер необхідно більше детально дослідити всі механіки відеоігор в даному жанрі.

Слід розпочати з того, що гра буде багатокористувацькою онлайн ареною, згідно з назвою жанру. Оскільки гра буде мультиплеєрною (онлайн), програма буде поділеною на дві частини: серверну та клієнтську, причому

клієнт не може спілкуватись з іншим клієнтом напряму, а лише завдяки серверу. Така модель дозволяє запобігти шахраюванню в грі, наприклад відправлення іншому гравцеві неправильного місця розташування. Весь ігровий процес буде відбуватись на сервері, а гравці будуть підключатись до нього. Для того, щоб гравець зміг управляти своїм героєм необхідно забезпечити механізм передачі інформації з клієнтської сторони на серверну. Найкраще для цієї мети підійде RPC (англ. Remote Procedure Call, Виклик віддалених процедур) – технологія, що дозволяє програмі на одному комп'ютері запускати функції (процедури) програми на іншому комп'ютері. Завдяки такому підходу, можна забезпечити відправлення на сервер дані від гравця, наприклад дані від ігрових контролерів про те які клавіші гравець натискає та що повинно відбуватись. Дана система буде працювати наступним чином:

1) Гравець натискає клавішу для пересування персонажу

2) Посилається запит на сервер про пересування гравця

3) Сервер перевіряє чи може гравець пересуватись в даному напрямку

4) Сервер переміщає персонажа або ні в залежності від пункту 3.

5) Нове місце розташування передається усім іншим гравцям в тому числі і гравцю, що відправляв запит.

Для пришвидшення даної технології і зменшення відчуття проблем з мережею можна переміщувати клієнт може переміщати локально, але все ж сервер буде перезаписувати дані про переміщення, щоб запобігти шахрайству. Отож, необхідно обрати ігровий рушій з мережевим кодом, який зможе підтримувати багато гравців одночасно.

Також, для ігор в жанрі MOBA характерне розміщення камери Top Down (вид зверху вниз), що означає що гравець буде бачити територію не лише перед своїм героєм, а й за ним. Згідно з оглядом в першому розділі було вирішено, що геймплей Dota 2 занадто затягнутий, а система BO5 є не

доцільною для не рейтингових ігор, оскільки можна зіграти один довший матч за 15 хвилин а не 5 коротких. Саме тому було вирішено використати режим «Deathmatch» - бій на смерть, вид геймплею багатокористувацьких ігор, основною задачею якого стоїть в безпосередньому знищенні супротивників (інших гравців). При цьому в даному режимі немає союзників чи інших цілей. Я вважаю, що даний режим найбільш підходящий під назву жанру МОВА, і разом з мапою невеликого розміру зможе забезпечити насправді «бойову арену», яка фігурує в назві жанру. Тривалість матчу буде приблизно 10 хвилин для оптимальності часу. Гравець після обирання героя відроджується в спеціальних зонах на мапі і після цього може вступити в бій. В кожного героя буде по 5 унікальних вмінь. Заради ігрового балансу кількість очок здоров'я та енергії буде в кожного персонажа різною. Енергія буде витрачатись на використання вмінь, але буде відновлюватись з часом. Також, буде доданий режим «Team Deathmatch» - режим, який повністю повторює попередній, але при цьому гравець буде працювати в команді і битись проти іншої команди. Перемога буде присуджена тому гравцеві чи команді, яка отримає найбільше очок за знищення своїх суперників.

Для даних ігор притаманна присутність AI, який можуть викликати герої своїми вміннями, або при підготовці до битви з справжніми гравцями пройти курс підготовки, який ознайомить гравця з геймплейними механіками, змагаючись проти істот з штучним інтелектом. При такому розкладі в гру слід обрати ігровий рушій, що підтримує систему для створення штучного інтелекту. В сучасних комп'ютерних іграх існує новітній інтерфейс користувача, що є інтуїтивно зрозумілим і повинен допомогти гравцеві розібратись в механіках гри. Оскільки, мої навички як художника та дизайнера зводяться до мінімуму, було вирішено обрати рушій з системою, що допоможе створити зрозумілий інтерфейс, та має багату бібліотеку 3D асетів, для створення ігор.

ВИСНОВОК ДО РОЗДІЛУ 2

Стрімке зростання комп'ютерних ігор у сфери розваг призвело до створення спеціальних програм для їх створення – ігрових рушіїв. Дані технології призначені для того, щоб полегшити життя розробників комп'ютерних ігор, оскільки більшість проблем, що стосуються базових потреб для створення ігор вирішені всередині самого рушія і дають можливість програмісту зосередитись на розробці ігрової механіки, а не на взаємодії з апаратним забезпеченням.

Більш детальний аналіз комп'ютерних ігор в жанрі МОВА дав можливість спроектувати модель ігрових процесів які повинні бути в майбутній грі, та як вони повинні гравець повинен взаємодіяти з іншими гравцями. На базі цього аналізу, з'явились певні критерії до ігрового рушія, на якому буде створена ця гра.

РОЗДІЛ 3
РОЗРОБКА СИСТЕМИ.

3.1 Вибір технологій

3.1.1 Вибір ігрового рушія

Враховуючи всі поставлені вимоги до гри, всі плюси та мінуси ігрових рушіїв, що були описані в пункті 2.2 даної дипломної роботи, а також мій досвід роботи з Unreal Engine 4 мій вибір припав саме на нього. Даний GE має наступні переваги, що дозволять пришвидшити розробку програми: система Blueprints, система для створення AI, мережевий код, що базується на технології RPC (Remote Procedure Calls) та Replication (реплікація – копіювання даних з серверу на клієнт з їх перезаписом), система для створення UI (User Interface). Також, в UE4 присутні й інші досить цікаві та корисні системи, як наприклад система створення матеріалів, що дозволяє налаштовувати зовнішній вигляд персонажів, мапи або ж інтерфейсу користувача. Всі елементи, ігрового рушія представлені в вигляді об’єктів, які мають набір характеристик. Слід зазначити, що ігровий контент – графіка (моделі персонажів, зброї, текстури для UI та моделі, що використовуються на мапах в грі), звуки - оригінальні і ліцензовані, взяті для цього проекту з онлайн-сервісу Unreal Marketplace. Отож, обравши даний GE я отримую велику перевагу і зменшую час розробки своєї гри до мінімуму, оскільки всі необхідні мені компоненти в ньому присутні.

3.1.2 Вибір платформи

Під ігрову платформу було вирішено обрати ПК, оскільки існує можливість одразу провести тестування готового результату. Обмежуватись якось конкретною ОС не має сенсу, так як UE4 підтримує кроскомпіляцію під різні операційні системи. Також, враховуючи статистику найбільш популярного онлайн-сервісу з цифрового розповсюдження комп’ютерних ігор та програм, що був розроблений та підтримуваний компанією Valve, Steam

кількість користувачів, що використовують OS X та Linux для того, щоб грати в комп'ютерні ігри зросла за квітень 2020 року.

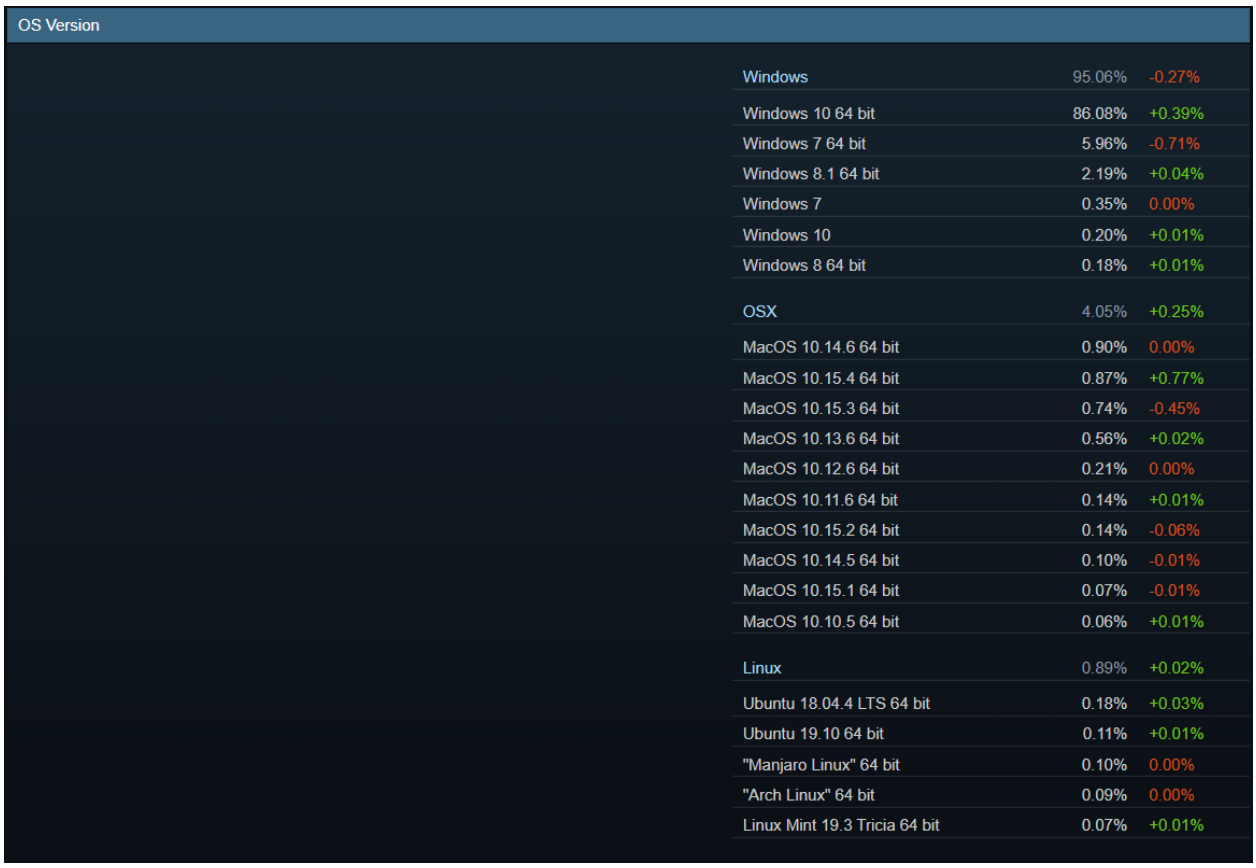


Рисунок 3.1 – Статистика використання різних ОС надана сервісом Steam за квітень 2020 року

Як показує статистика більшість геймерів користуються ОС сімейства Windows (95.06%), 4.05% користуються системами сімейства OS X і лише 0.89% грають в відеоігри використовуючи операційні системи сімейства Linux. Саме тому я не бачу потреби обмежуватись якось ОС і тим самим не давати можливості усім гравцям спробувати зіграти в мою гру, не використовуючи допоміжні засоби.

3.1.3 Вибір мови програмування та IDE

Оскільки, ставка, як на основний засіб для розробки, була зроблена на Unreal Engine 4, то і використовуватись для представленої системи буде мова програмування C++ та система візуального програмування Blueprints. В

даному рушії відсутній текстовий редактор, що унеможливлює написання коду на C++ прямо в Unreal Editor. Саме для цього необхідно вибрати середовище розробки для написання та компіляції коду. UE4 за замовчуванням підтримує розробку програм використовуючи Microsoft Visual Studio.

Microsoft Visual Studio – інтегроване середовище розробки від компанії Microsoft, в якому існує можливість створювати як консольні застосунки так і з графічним інтерфейсом. Воно включає в себе редактор коду з підтримкою IntelliSense – технології створеної компанією Microsoft, задача якої в автодоповненні назв функцій при вводі початкових букв, доступі до документації і знищення неоднозначності в назвах змінних, функцій та методів. Також, є вбудований дебагер, що дозволяє налагоджувати код. Існує можливість створювати, та додавати сторонні плагіни для розширення функціоналу IDE. Visual Studio пропонується в трьох версіях:

- Visual Studio Community – безкоштовна версія IDE для студентів і незалежних розробників.
- Visual Studio Professional – платна версія з професійними інструментами для розробки для малих команд.
- Visual Studio Enterprise – платна версія з професійними інструментами для продуктивної роботи та координації команд розробників будь-якого розміру.

Для даного проекту буде використовуватись Visual Studio 2019 Community, оскільки даної безкоштовної версії досить для створення комп’ютерної гри. Для цього буде використана частина даного IDE – Visual C++, а зокрема MSVC v142 – VS 2019 C++ x64\x86 build tools (v14.24) (набір компілятора та бібліотек C++) та для розробки під ОС Windows 10 – Windows 10 SDK (10.0.18362.0). Дані компоненти додаються до IDE після того, як в меню оновлення обирається пакет Game development with C++.

Основною мовою програмування в Unreal Engine 4 являється модифікація C++, вона більш заточена під потреби ігрового рушія, має декілька нових типів даних та специфікаторів, що дозволяють створювати змінні та функції, відкриті для редагування в технології Blueprints. Загалом в UE4 багато нових типів даних та структур, я опишу ті, які були використані при написанні програми.

Новим типом змінних являються нові контейнери, що заміняють STL-контейнери:

- TArray<type> - динамічний масив, працює як STL- контейнер vector. Окрім типу даних можна також, задавати власний алокатор пам'яті під цей масив. Може бути використаний при написанні багатокористувацьких онлайн ігор, оскільки може бути реплікований.
- TMap<type, type> - контейнер, заміна STL-контейнеру map. Параметрами приймає два типи даних, що будуть зберігатись як ключ – значення. Як і в TArray можна задати як параметр алокатор пам'яті, але даний тип не реплікується в мережі, тому при використанні його в онлайн-іграх можуть виникнути певні труднощі.
- TSet<type> - заміна STL- контейнеру set. Зберігає унікальні елементи без їх сортування. Як і TMap не може бути реплікованим по мережі.
- TSubclassOf<type> - шаблонний клас, що дозволяє забезпечити безпеку типів UClass. Даний клас може набувати значення лише того типу або дочірніх від нього, який буде наданий в поле type.

Також, використовуються наступні структури:

- FVector – структура, що являє собою вектор в тривимірному просторі і складається з компонентів із плаваючою комою. Зберігає в собі координати X, Y, Z та підтримує різні операції над тривимірними векторами.

- FRotator – структура, що зберігає в собі поворот об’єкту, складається з компонентів плаваючою комою та зберігає три повороти: Pitch (поворот по осі X), Yaw (поворот по осі Y), Roll (поворот по осі Z). Підтримує різні операції, що необхідні для розрахування повороту.
- FTransform – використовується для зберігання місця розташування (FVector), повороту (FRotator) та його масштабування (FVector).
- FTimerHandle – унікальний дескриптор, що використовується для розрізнення таймерів з однаковими делегатами. За допомогою цього дескриптору можливо скидати таймери.
- FHitResult – структура, що використовується для зберігання результату попадання при трасуванні об’єктів.

Слід також зазначити, що всі C++ класи, що будуть створені повинні бути нащадками вже готових класів, вбудованих в UE4.

Наступною системою, що буде використовуватись для написання даної дипломної роботи являється технологія Blueprints. Слід почати з того, що всі Blueprint-класи, що створюються в редакторі являються нащадками вбудованих в UE4 C++ класів, або класів, що можна створити власноруч, тобто він буде володіти всіма властивостями і функціоналом, що і класи-предки. Як було сказано раніше програмування відбувається графічно, тобто будується алгоритм за допомогою спеціальних вузлів-блоків, що містять в собі код C++. Завдяки такій технологій при остаточній компіляції та пакуванні програми існує можливість нативізувати Blueprint-код, тобто перегенерувати на C++, що збільшить його швидкодію. Саме програмування зводиться до створення функцій та побудови графу, носить назву Event Graph (Граф подій), який показує як саме повинен діяти той чи інший об’єкт. [9]

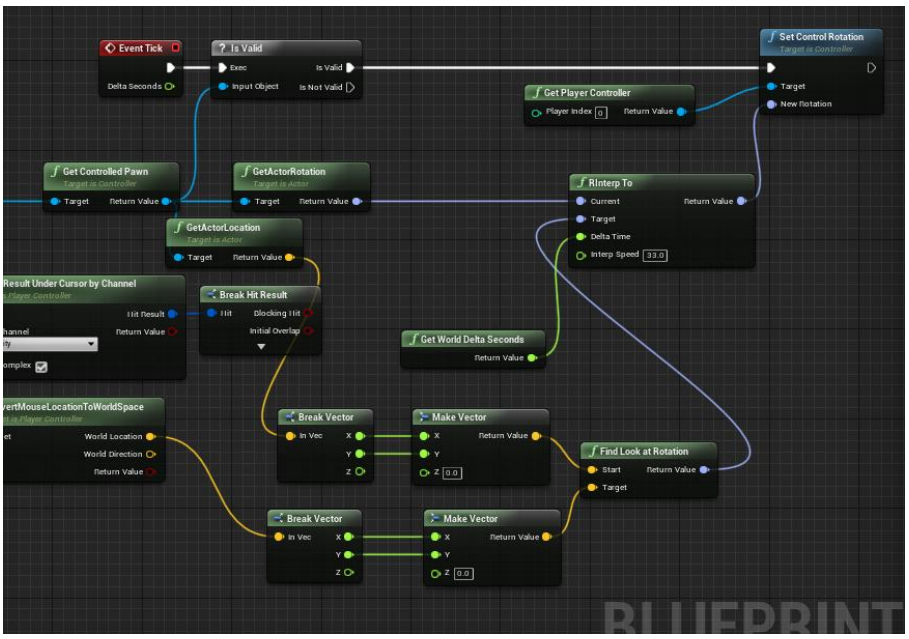


Рисунок 3.2 – Частина Event Graph

Самі блоки поділяються на декілька видів, основними з яких є функції (сині та зелені блоки на рис. 3.2) та Events (події, червоний блок на рис. 3.2). Функції, що не можуть бути включені у логічний ланцюжок виконання і не несуть змін до об’єкту називаються Pure-функціями (чистими функціями, зелені блоки на рис. 3.2, їх основною задачею являється прораховування певних формул і повернення результатів, без зміни якихось даних об’єкту). Події та звичайні функції взаємодіяти одна з одною, але між ними є різниця. Наприклад, існують базові події, що викликаються самим рушієм і за допомогою них можна запрограмувати ті чи інші дії. Також, лише події можуть мати RPC специфікатори, що дозволяє взаємодію клієнта та серверу.

В C++ створення подій та функцій це звичайні C++ функції, але з різними специфікаторами, що передаються в макроси, мова про які піде далі. Та перед цим необхідно зазначити, що оскільки Blueprint-класи є нащадками C++ класів, то властивості в них залишаються такими самими, або ж можуть бути розширеними. Початкові ж дані розміщені у вікні Class Defaults і знову в залежності від специфікаторів, що були описані в C++ можуть бути доступними або ні у цьому вікні. [10]

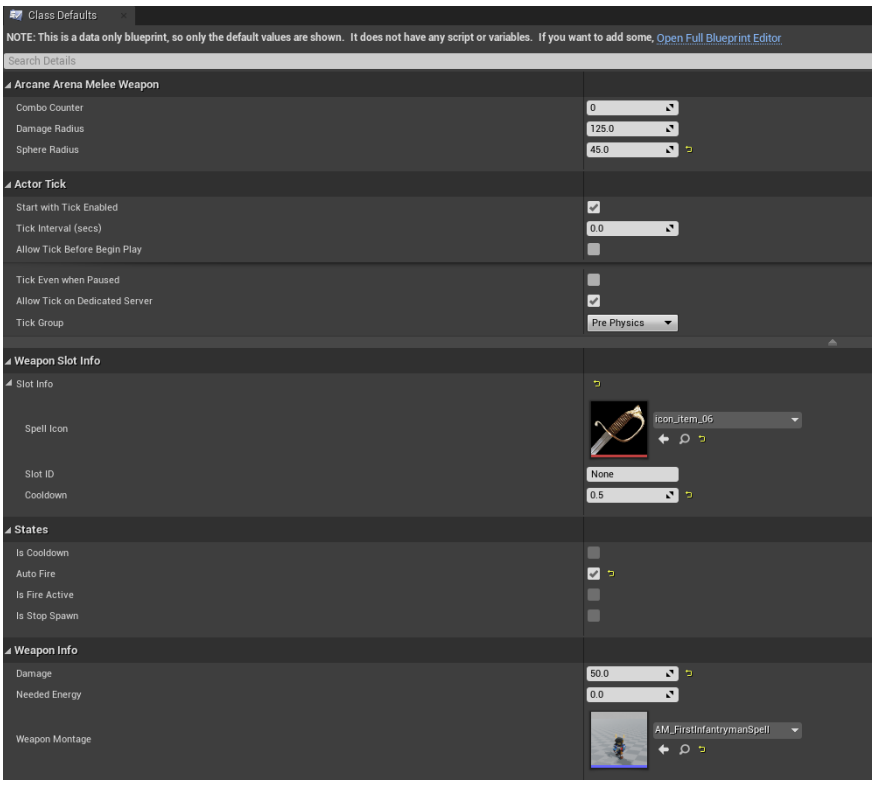


Рисунок 3.3 – Приклад вікна Class Defaults та змінних, що описані в C++ і доступні для редагування в дочірніх класах Blueprint

Оскільки в розробці буде використовуватись технологія Blueprints необхідно розібрати специфікатори, що дозволяють налаштовувати дані та викликати функції визначені в C++, в Blueprints. Для того, щоб дозволити редагувати змінні в Blueprint-класах необхідно перед оголошенням змінної в C++ необхідно прописати макрос UPROPERTY(), тобто код буде виглядати наступним чином: [14]

```
UPROPERTY (EditDefaultsOnly, BlueprintReadOnly, Category = ArenaWeapon")
    float Damage;
```

В якості параметрів макрос приймає спеціальні специфікатори, що прописані в UE4, їх доволі багато та в дипломній роботі використовуються лише деякі з них: [14]

- VisibleAnywhere – специфікатор, що дозволяє побачити у всіх вікнах властивостей Blueprint-класу (як у вікні базового класу, так і у вікнах екземплярів даного класу на мапі). Також, існують версії VisibleDefaultsOnly та VisibleInstanceOnly, що дозволяють приховати

змінну в вікнах властивостей екземплярів об’єкту та базового класу відповідно. Редагувати змінну використовуючи ці специфікатори у вікнах властивостей неможливо і він не сумісний з специфікаціями типу «Edit».

- EditAnywhere - специфікатор, що дозволяє побачити у всіх вікнах властивостей Blueprint-класу (як у вікні базового класу, так і у вікнах екземплярів даного класу на мапі). Також, існують версії EditDefaultsOnly та EditInstanceOnly, що дозволяють приховати змінну в вікнах властивостей екземплярів об’єкту та базового класу відповідно. Даний специфікатор дозволяє редагування в вікнах властивостей та є несумісним з специфікатором типу «Visible».
- BlueprintReadWrite – специфікатор, що дозволяє читання змінної та запис даних в неї в самому коді Blueprint. Існують дві інші модифікації BlueprintReadOnly, BlueprintWriteOnly, що дозволяють лише читати або лише записувати змінну відповідно.
- Category – специфікатор, що дозволяє відсортувати змінні та функції по власним створеним категоріям, що будуть відображатись у редакторі Blueprint графів.
- BlueprintAssignable – використовується лише для делегатів багатоадресної передачі (MULTICAST_DELEGATE). За допомогою цього специфікатора можна підв’язати виконання якоїсь функції в Blueprint-коді до цього делегату. Простіше кажучи, MULTICAST_DELEGATE являє собою механізм синхронізації та оповіщення про те, що якась подія відбулась.
- AllowPrivateAccess – дозволяє доступатись та редагувати змінні в Blueprint класах, що були оголошені в private секції класу-предку.
- Replicated – специфікатор, що використовується в мережевих іграх та коді. Він дає зрозуміти серверу, що дану змінну необхідно копіювати з клієнту на сервер.

- ReplicatedUsing – специфікатор, який також використовується в мережі і, на відміну від Replicated дозволяє викликати якусь функцію, що була підв’язаною до нього під час зміни даної змінної в мережі. Функція викликається на всіх клієнтах та сервері, що допомагає вирішити деякі проблеми при програмуванні мультиплеєрних ігор. Приклад використання: ReplicatedUsing= OnSpellCreated, де OnSpellCreated – назва функції, що створена в даному класі, причому обов’язково повинен бути присутній макрос UFUNCTION(), мова про який піде наступною.

Як було сказано раніше, Blueprint-код будується використовуючи спеціальні вузли-блоки, що всередині містять C++ код, і працюють як функції в даній мові програмування. Саме тому використовуючи макрос UFUNCTION() можливо створювати власні блоки та викликати їх в коді Blueprint. Приклад використання даного макросу: [15]

```
UFUNCTION (BlueprintCallable, Server, Reliable, WithValidation, Category =
"ArcaneArenaBaseWeapon")
    virtual void Attack (FVector SpawnLocation);
```

При створенні дипломного проекту я використовував наступні специфікатори для створення власних Blueprint-блоків: [15]

- BlueprintCallable – дає можливість викликати дану функцію в Blueprint-графі класу або рівня(мапи).
- BlueprintPure – даний специфікатор означає неможливість зміни об’єкту, для якого вона викликається. Зазвичай використовується для доступу до інформації (прикладом таких функцій можуть бути getter – функції для різних класів).
- Server – специфікатор, що використовується для RPC-функцій. Використовується клієнтом для того, щоб послати запит на сервер для обробки інформації на ньому та виконанні певних дій.

- NetMulticast – одна з RPC-функцій, що викликається з серверу. Використовується для багатоадресної передачі всім клієнтам, а також викликається на самому сервері.
- Client – остання RPC-функція викликається з серверу на певному клієнті і виконується лише на ньому.
- Reliable – параметр, що використовується для забезпечення виконання RPC-функції незалежно від мережевих проблем.
- WithValidation – означає необхідність створити додаткову функцію до основної, лише додаючи до назви в кінці _Validate, що повертає значення типу bool. Дана функція перед має ті самі вхідні параметри, що й основна і перед викликом головної функції, перевіряє на правильність вхідні параметри і чи може основна функція бути виконаною. В онлайн іграх, якщо _Validate функція повертає false, система розпізнає гравця як нечесного і відключає від сеансу гри. Приклад використання даної функції:

```
void AArenaBaseWeapon::UseWeaponServer_Implementation (bool bIsUse)
{
	UseWeapon(bIsUse);
}
bool AArenaBaseWeapon::UseWeaponServer_Validate (bool bIsUse)
{
	return true;
}
```

- Category – працює так само як і в UPROPERTY.
- BlueprintImplementableEvent – функція, що не має програмного коду на C++ і повинна бути перезаписана в Blueprint-класі, але може викликатись в коді C++. Корисна, коли для декількох Blueprint-нащадків класу програмна реалізація цієї функції може бути різною і її необхідно перезаписати в самому Blueprint-класі. Також існує функція BlueprintNativeEvent яка має програмну реалізацію в C++, але може бути перезаписаною в Blueprint-коді, з викликом чи без батьківської логіки.

3.2 Основні рішення з реалізації комп'ютерної гри та її компонентів

3.2.1 Конфігурація компонентів додатку

Усі Blueprint-класи, сирцеві файли, асети проекту розгруповані відповідно до їх функцій та призначення. Для підключення додаткових залежностей та пакетування проекту використовується файл ArcaneArena.Build.cs, також під різні версії проекту (версія для редактора, версія для серверу і т.д.) використовуються файли з назвою, що закінчуються на Target.cs.

3.2.2 Основні класи даного проекту

Оскільки в Unreal Engine 4 підтримує власну систему для створення багатокористувацьких онлайн ігор, необхідно розбити реалізацію додатку по класам в залежності від того як класи взаємодіють при клієнт-серверній взаємодії.

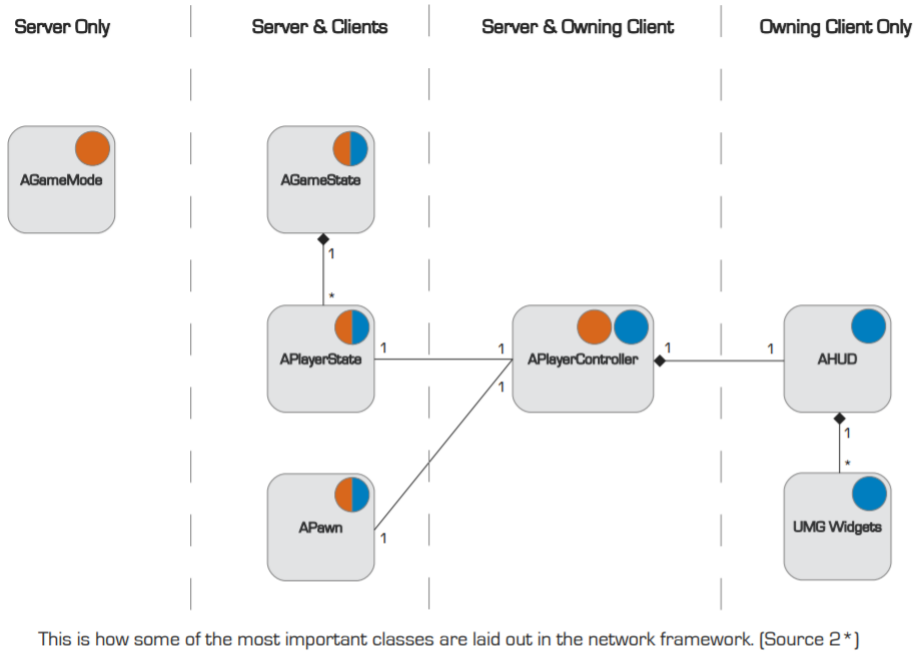


Рисунок 3.4 – Взаємодія основних класів UE4[12]

Клас ArenaBaseGameMode

Даний клас являється дочірнім класом від ABaseGameMode і його екземпляр існує лише на серверній стороні гри і не може бути змінений на стороні клієнту. Для спілкування з даним класом необхідно використовувати RPC-функкції. За допомогою нього задаються основні правила гри та інші основні класи гри. Система побудована таким чином, щоб під кожний GameMode можливо обрати різні базові класи, а під кожен рівень (мапу) можна обрати свій GameMode. Таким чином ми отримуємо дуже швидку систему для налаштування різних мап.

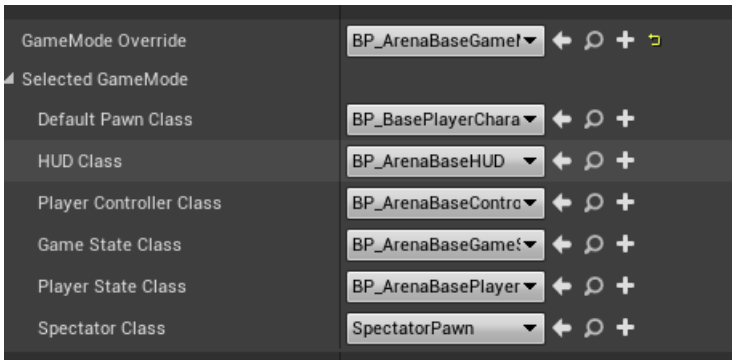


Рисунок 3.5 – Основні класи для GameMode

Основні методи, що реалізує даний клас:

- virtual bool PlayerCanBeDamaged (T* DamgedPlayer, T* IntigatedBy) – метод, що бути перевизначений в дочірніх класах, перевіряє можливість нанесення пошкоджень гравцеві. Як параметри приймає контролери гравців, того, що наніс пошкодження та кому вони були нанесені, де T – AArenaBaseController.
- void UpdatePoints (T* DeadPlayerController, T* InstigatorController) – метод, що оновлює статистику гравцям (кількість смертей, та кількість переможених гравців). Параметри – це контролери гравців, того, що наніс пошкодження та кому вони були нанесені, де T – AArenaBaseController.

- void SpawnPlayer (AArenaBaseController* PlayerController, TSubclassOf<AArenaBaseCharacter> Class) – викликається для породження герою на сервері та «вселення» в нього контролеру гравця. Параметрами є сам контролер та клас героя, що був обраний гравцем.
- void OnCharacterDead (T* DeadPlayerController, T* InstigatorController) – викликається при смерті персонажа гравця. Виконує комплексні дії на серверній стороні. Параметрами є контролери гравців, де T – AArenaBaseController.
- void OnTimeLimitReached () – метод викликається коли ліміт по часу був вичерпаний. Викликає метод StopMatch з параметром EStopReason::TimeLimit.
- void OnChooseCharacterClass (AArenaBaseController* PlayerController, TSubclassOf<AArenaBaseCharacter> Class) – метод, що на підв’язується до делегата багатоадресної передачі. Після того як отримує сигнал про те що гравець обрав свого героя запускає процес породження цього персонажа на сервері.
- virtual void BeginPlay () override – базовий метод рушія UE4, викликається коли екземпляр класу був створений. Підв’язує метод OnChooseCharacterClass() до делегату багатоадресної передачі.
- void StopMatch (EStopReason Reason) - Відбувається очищення мапи та виклик делегату на який підв’язана функція початку голосування за нову мапу. Параметром слугує змінна типу enum, що вказує на причину припинення матчу.
- virtual AArenaPlayerSpawnZone* GetSpawnZone () – оскільки гра підтримує декілька режимів, то залежно від нього вона буде повертати зону в якій може бути породжений герой гравця. Якщо встановлений командний режим то зона буде відібрана відповідно до команди гравця, інакше буде видана будь-яка зона.

Клас AArenaBaseCharacter

Даний клас являється нащадком від класу ACharacter і предком для всіх класів персонажів, зокрема для BP_BasePlayerCharacter, що слугує базовим персонажем (Default Pawn Class) для ігрових режимів. За замовчуванням при підключенні гравця, GameMode породжує персонажа класу Default Pawn Class та вселяє в нього контролер гравця. Даний клас присутній та доступний як на стороні клієнта так і на стороні серверу. Враховуючи те, що контролер гравця пересувається по різному в залежності від того в якого персонажа він поміщений, то в даному класі описана логіка переміщення героя, а також додані класи-компоненти для відслідковування кількості очок здоров'я та його готових вмінь.

Клас містить в собі наступні методи:

- virtual void PawnClientRestart () override – вбудований метод UE4, викликається на клієнтській стороні після того, коли в персонажа був вселений контролер.
- virtual void BeginPlay () override – метод UE4, що викликається коли об'єкт був створений, на стороні серверу викликає запуск створення вмінь героя.
- void MoveAlongX (float Value) – використовується для переміщення по осі X.
- void MoveAlongY (float Value) – використовується для переміщення по осі Y.
- void BindActionBarSlot (bool bIsUse, FName SlotID) – використовується для взаємодії з інтерфейсом користувача, в даному випадку підсвічує використане вміння героя при натисканні відповідної клавіші.
- void On_RepCharacterState () – метод, що підв'язаний до змінної CharacterState. Викликається на сервері та кожному клієнті коли ця

змінна змінює свій стан в мережі (ReplicatedUsing). В залежності від стану героя виконуються ті чи інші дії (наприклад при смерті відключається здатність пересуватись і реагувати на ввід користувача).

- void UpdateCharacterState (EPlayerState NewState) – метод, що може викликатись як на клієнтській так і на серверній стороні. Перевіряється якщо, виклик відбувається на клієнті то даний метод викликає RPC-функцію UpdateCharacterStateServer і після цього змінює CharacterState на клієнті для пришвидшення обробки інформації.
- void UpdateCharacterStateServer (EPlayerState NewState) – RPC-метод, що викликається на серверній частині і обновляє змінну CharacterState.
- void UpdateBarClient (FName BarID, float NewPercentage) – RPC-метод викликається з серверної частини на клієнта, якому належить цей об’єкт. Використовується для взаємодії з UI.

Клас AArenaBaseController

Даний клас є нащадком APlayerController та предком для BP_ArenaBaseController (займає місце Player Controller Classв налаштуваннях ігрового режиму). Даний клас доступний лише на серверній стороні та клієнті, який отримав цей контролер. Контролер видається один на всю гру після того як гравець був підключений до матчу.

В ньому реалізовані наступні методи:

- void UpdateCharacterClass (TSubclassOf<AArenaBaseCharacter> NewCharacterClass) – якщо виклається на клієнтській стороні, то відбувається виклик UpdateCharacterClassServer, на серверній стороні відбувається оновлення змінної, що відповідає за клас героя відповідно до вибору гравця.

- void UpdateCharacterClassServer (TSubclassOf<AArenaBaseCharacter> NewCharacterClass) – викликає UpdateCharacterClass на серверній частині.
- void SetVoteServer (int MapIndex) – RPC, що відсилає голос гравця за наступну мапу на сервер.
- void CreateHUD () – створює UI гравця після того як його контроллер був вселений в обраного героя.
- void DisplayDamage (AActor* DamagedActor, float Damage) – викликається на клієнті, що наніс пошкодження певному гравцю, взаємодіє з інтерфейсом користувача, виводить на екран кількість втрачених очок здоров'я, поблизу гравця, що втратив їх.
- virtual void BeginPlay () override – вбудована функція UE4, на серверній стороні відбувається підв'язка до MULTICAST_DELEGATE, що відповідає за закінчення матчу.
- void OnMatchEnded (EStopReason Reason) – викликається після того, як був отриманий сигнал про закінчення матчу, обнуляється таймер перероджження, щоб запобігти перероджження персонажа після закінчення матчу.
- void Respawn () – метод, що викликається на сервері, використовується для перероджження персонажу
- virtual void Tick (float DeltaTime) override – вбудований метод UE4, що викликається кожен кадр під час ігрового процесу. Викликає метод RotateCharacterToMouse.
- void LaunchRespawnTime (float RespawnTime) – викликається на сервері після смерті персонажа. Запускає таймер після якого відбувається перероджження гравця в нового героя.
- void RotateCharacterToMouse () – розвертає героя, якщо він існує до того місця на яке наведений курсор миші гравця.

Клас AArenaBaseGameState

Клас відомий як серверу так і всім гравцям. Використовується для відображення та зміни глобального стану гри. Містить в собі скільки часу пройшло з моменту початку матчу, компонент голосування за наступну мапу та делегати багатоадресної передачі, що відповідають за закінчення, початок матчу та вибору гравцем нового героя.

- void TryToStartRound () – викликається на сервері, після того як контролер гравця був вселений в обраного персонажа. Перевіряється чи достатньо на мапі гравців, щоб розпочати бій, якщо так відбувається зміна змінної MatchState.
- void ClearMatchTimer () – викликається після завершення матчу, таймер часу обнуляється, щоб не перейти в від'ємний час.
- bool IsEnoughPlayersToStart () – перевірка чи достатньо гравців було підключено до серверу, щоб розпочати гру.
- void GetMatchTime (int &MinutesOut, int &SecondsOut) – повертає час матчу в хвилинах та секундах, використовується в інтерфейсі користувача.
- void PlayMatchTimer () – метод, що викликається кожну секунду, обраховує час, що залишився до кінця матчу.
- void On_MatchStateChanged () – ReplicatedUsing метод, який викликається при зміні MatchState по мережі.
- void OnMatchEnded () – метод підв'язаний до MULTICAST_DELEGATE кінця матчу, запускає початок голосування за нову мапу, підв'язує нові делегати, що використовуються для закінчення голосування.
- void OnVoteEnded () – кінець голосування, виклик механізму переходу на карту, що перемогла в голосуванні.

- virtual void BeginPlay () override – базова функція, запускає таймер, що відповідає за час матчу, підв’язує функції інтерфейсу користувача до делегату кінця раунду.
- void ServerTravelToMap (FName Map) – виконує консольну команду переходу до нової мапи.
- void MatchEndedMulticast (EStopReason Reason) – RPC-метод, що викликається на всіх клієнтах та сервері під час закінчення матчу. Викликає оповіщення завдяки делегату про те, що матч закінчився.

Клас ArenaBasePlayerState

Даний клас присвоюється кожному гравцеві і зберігається на протязі всього матчу на даній мапі. Зберігає в собі стан гравця та змінні, що відповідають за його кількості очок. Даний клас доступний як на серверній стороні так і на клієнтській. З даним класом може напряму взаємодіяти клас AArenaBaseGameState та AArenaBaseController в той час як сам клас може взаємодіяти напряму лише з останнім.

Реалізовані наступні методи:

- void AddDeathPoint () – метод, що підвищує кількість смертей на одиницю;
- void AddKillPoint () – підвищення кількість очок за знищення інших героїв на одиницю;
- void GetPlayerStats (int& OutKills, int& OutDeaths) – повертає значення кількості смертей та кількості знищень ворогів. Використовується в інтерфейсі користувача для виводу на екран цих даних.
- virtual void ClientInitialize (AController* C) override – базовий метод UE4, викликається коли PlayerState реплікується на клієнт і стає валідним. Викликається лише в логіці гравця, якому належить цей клас. Використовується для ініціалізації інтерфейсу користувача та підв’язки інтерфейсу до делегатів, що відповідають за стан героя.

- void OnChangeAliveMulticast (bool NewState, AArenaBaseContorller* DeadPlayerController, AArenaBaseController* InstigatorController) – RPC-метод, що викликається на всіх клієнтах та сервері під час зміни стану персонажу, що знаходиться в даному класі. Викликає делегати, що відповідають за зміни стану живий/мертвий.

Клас AArenaBaseHUD

Даний клас є останнім з основних класів, які піддаються налаштування в ігровому режимі. Відповідає за виведення інтерфейсу користувача на екран монітору. Доступний лише на стороні того клієнту для якого був створений і доступ до даного класу можна отримати лише через контролер гравця. З серверної частини неможливо доступитись до цього класу.

Реалізовано наступні методи:

- void OnInitialize () – метод, що викликається в класі ArenaBasePlayerState на в методі ClientInitialize. Даний метод виводить на екран вибір герою та підв’язує функції на делегати стану персонажа живий/мертвий з класу ArenaBasePlayerState, для виведення різних віджетів в залежності від стану персонажа.
- void OnMatchEnded (Estop Reason) – метод, що підв’язується до делегату кінця матчу з класу AArenaBaseGameState. Використовується для виведення кінцевого рахунку та переможця гри, а також віджету голосування за наступну мапу.
- void DisplayDamage (AActor* DamagedActor, float Damage) – метод, виводить нанесені пошкодження для Damaged Actor. Даний метод викликається в AArenaBaseController в методі DisplayDamage.

  Наступні методи реалізовані використовуються для виведення/приховування на екран монітору різні види віджетів:

- void ToggleHUD (bool bShow) – вивід на екран HUD (head-up display) – частина інтерфейсу користувача, що дозволяє отримати різну інформацію на викликаючи інші додаткові меню. Виводить

інформацію про здоров’я, енергію, вміння героя та час, що залишився до кінця матчу.

- void ToogleChooseCharacter (bool bShow) – вивід на екран меню вибору героя.
- void ToggleSpectator (bool bShow) – вивід на екран віджету глядача, що з’являється коли герой гравця помирає.
- void TogglePostMatch (bool bShow) – вивід меню голосування та кінцевої статистики всіх гравців.
- void ToggleGameStats (bool bShow) – вивід проміжної статистики під час матчу всіх гравців.

Також, були розроблені додаткові класи, необхідні для ігрового процесу, такі як UArenaHealthComponent – клас-компонент, що підключається до AArenaBaseCharacter, використовується для обрахування кількості очок здоров’я та енергії та їх менеджменту. Даний клас містить наступні методи:

- void RegenerateHealth (float NewHealth) – метод, що використовується для відновлення життєвих показників персонажа гравця. В ньому існує вбудована перевірка, що не дозволяє гравцеві шахраювати та відновити кількість очок здоров’я більше, ніж було на початку.
- float GetEnergy () – викликається для отримання інформації про енергію гравця.
- void GetHealthStats (float &HealthOut, float &MaxHealthOut) – метод для отримання інформації про життєві показники героя. Використовується в інтерфейсі користувача.
- void GetEnergyStats (float &EnergyOut, float &MaxEnergyOut) метод для отримання інформації про енергію героя. Використовується в інтерфейсі користувача.
- void ApplyDamage (float NewDamage) – метод, що викликається при нанесенні пошкоджень персонажу гравця. Віднімає необхідну

кількість очок здоров’я та перевіряє чи залишається живим герой після нанесення пошкоджень.

- void OnPlayerTakeAnyDamage(AActor* DamagedActor, float Damage, const class UDamageType* DamageType, class AController* InstigatedBy, AActor* DamageCauser) – метод, що підв’язується на делегат отримання пошкоджень. Викликається після того, як персонаж для якого був створений даний компонент отримав пошкодження. Перевіряє чи гравець може отримати пошкодження в даний момент часу та викликає метод ApplyDamage.
- DisplayDamage (float Damage) – метод, використовується в інтерфейсі користувача для відображення пошкоджень, які наніс гравець іншим гравцям. Дозволяє інтуїтивно зрозуміти гравцеві чи його заклинання або якась навичка досягла своєї цілі.

UArenaWeaponManager – клас-компонент, також використовується в AArenaBaseCharacter, використовується для менеджменту вмінь героя, їх ЧІВ.

Даний клас містить в собі наступні методи:

- void OnSpellsCreated () – метод, викликається після того як всі вміння героя були створені, використовується для відображення вмінь в інтерфейсі користувача та їх підв’язки під певні клавіші, що задає користувач в налаштуваннях гри.
- void OnCharacterDead () – метод викликається при смерті персонажа, використовується для видалення з арени вмінь персонажа.
- void CreateSpells () – метод, що викликає сам герой після переродження. Створює вміння, в залежності від класу героя.

AArenaBaseWeapon – базовий клас вмінь кожного героя. Від даного класу повинні бути наслідувані всі вміння героїв для підтримки всього функціоналу. Містить наступні методи:

- void StopCooldown () – метод, що викликається після завершення

таймеру ЧІВ певного вміння. Перевіряє чи може бути викликане вміння ще раз і при позитивному результаті виконує повторний виклик.

- bool CanUse (bool bIsUsed) – перевіряє можливість виклику вміння героя. Поверне позитивний результат, якщо герой живий і знаходиться не в стані атаки.
- virtual void Use (bool bIsUsing) – віртуальна функція інтерфейсу IActionbarInterface, використовується для прив'язки слотів інтерфейсу користувача під вміння. При натисканні спеціальних клавіш гравець буде викликати дану функцію для вміння, що було підв'язано під слот, клавіша якого була натиснутою.
- virtual bool UseWeapon (bIsUse) – віртуальна функція, що викликається як на клієнті так і на сервері, відповідає за відображення виклику вмінь. Логіка даної функції поділена на клієнтську та серверну частини.
- void UseWeaponServer (bIsUse) – серверний варіант функції UseWeapon.
- virtual void Attack (FVectorSpawnLocation) – віртуальна функція, що використовується для виклику певної логіки вміння на сервері. Підв'язується під анімацію гравця.
- bool IsEnoughEnergy () – перевіряє чи достатньо енергії для використання цього вміння.
- void PlayAnimMontage () – метод, що викликає відтворення анімації на клієнті.
- void PlayAnimMontageMulticast () – серверний метод, викликає PlayAnimMontage на всіх клієнтах.
- void StopCooldownClient () – метод, використовується для інтерфейсу користувача. Графічно дає зрозуміти гравцеві, коли те чи інше вміння його персонажу готове для використання.

3.2.3 Графічний інтерфейс користувача

Ігровий рушій UE4 має вбудовану систему створення інтерфейсу користувача, що має назву UMG (Unreal Motion Graphics) – дозволяє створювати різні HUD, меню та інтерфейси для виводу інформації про ігровий процес, що називаються віджетами. Віджети також являються об’єктами в UE4 і мають спільний батьківський клас C++ - UUserWidget. Для створення даних віджетів використовується спеціальний редактор, вбудований в UE4. [16]

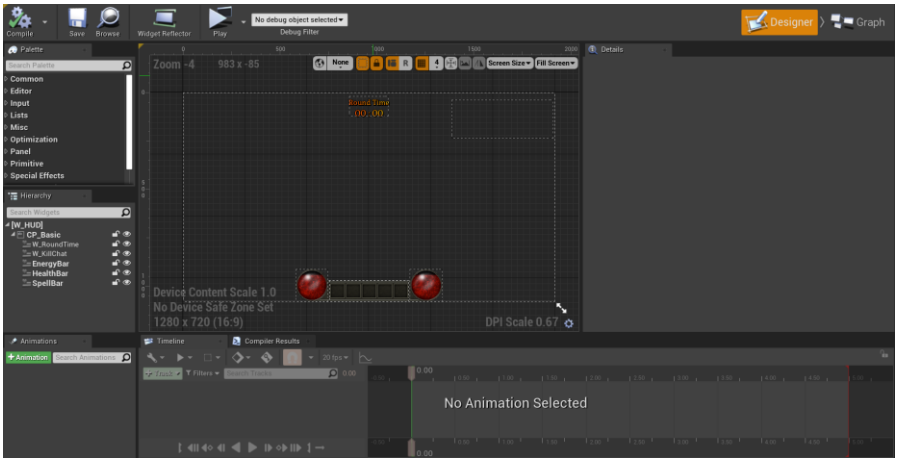


Рисунок 3.6 – Редактор для створення віджетів в UE4

На рис. 3.6 зображений редактор віджетів, та власне сам віджет HUD, що є основним в матчі. Він складається з декількох простіших підвіджетів, і така конструкція позволяє швидко змінювати його вигляд. Він виводить час до закінчення матчу, вміння та їх ЧІВ героя, за якого грає гравець, та його кількість очок здоров’я та енергії. Окрім самого редактор, існує вікно графу Blueprint, де відбувається програмування даного віджету на взаємодію з ігровим процесом. [16]

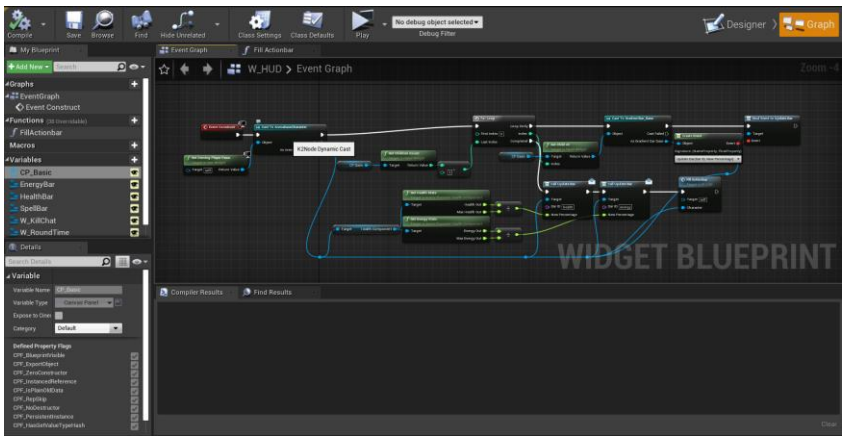


Рисунок 3.6 – Blueprint-граф віджету W_HUD

Також, гравці повинні змагатись між собою на деякій мапі. Саме тому використовуючи редактор Unreal Engine 4 була створена мапа, для тестування вмінь героїв та проведення тестового бою.

Рисунок 3.7 – Тестова мапа

Кінцевим результатом є гра в жанрі MOBA, що підтримує матч до 16 осіб, основним режимом якої є знищення своїх опонентів.

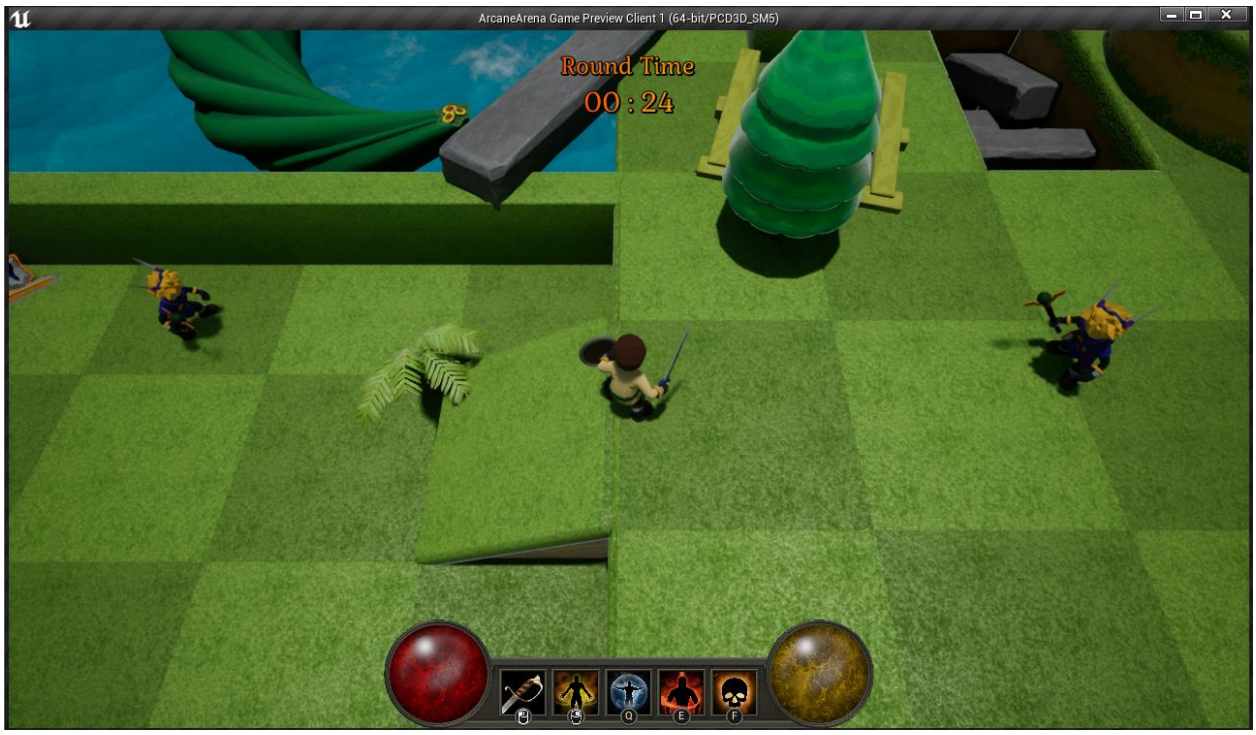


Рисунок 3.8 – Кінцевий результат

ВИСНОВОК ДО РОЗДІЛУ 3

В розділі було розглянуто аспекти вибору технологій для реалізації комп’ютерної гри, наведено докази для обґрунтування вибору ігрової платформи, мов програмування.

Розглянуто детально класи та методи реалізації програмного продукту, що дозволять побудувати гнучку та розширяєму архітектуру. Було розроблено гнучкий програмний продукт, який можна доповнювати та розширювати без значних затрат часу та готовий для продажу під ігрову платформу ПК під різні ОС.

ВИСНОВКИ

Даний дипломний проект присвячений розробці створення комп’ютерної гри в жанрі МОВА на основі існуючого ігрового рушія.

В ході роботи було розглянуто існуючі жанри відеоігор, існуючі рішення в даному жанрі та на скільки створення ігор є актуальною темою на сьогоднішній день. Були розглянуті геймплей ігор жанру МОВА, виділені основні недоліки ігрових механік, та порівняно їх реалізацію по відношенню одне до одного.

Було надано опис предметної області, визначені основні сутності, гри, що розробляється. Визначені основні функції та вимоги до готового продукту. Оскільки гра буде створюватись на базі існуючого ігрового рушія, був проведений детальний аналіз існуючих рушіїв, визначення їх плюсів та мінусів. Був обраний найбільш підходящий ігровий рушій – Unreal Engine 4, що містить в собі усі необхідні технології для реалізації гри.

Дана комп’ютерна гра має зрозумілий інтерфейс та прості ігрові механіки. Оскільки графіка не є надто реалістичною і використовуються низькополігональні моделі, може запускатись на комп’ютерах з не дуже дорогим апаратним забезпеченням. Використовуючи кроскомпіляцію гра не підв’язана під певну ОС та не обмежує можливість грати в неї гравцям з іншими операційними системами. Завдяки об’єктно-орієнтованості коду та модульності функціонал гри може бути легко розширений, шляхом додавання нових героїв або ж режимів ігрового процесу.

ПЕРЕЛІК ПОСИЛАНЬ

1. Unity User Manual [Електронний ресурс] – Режим доступу: https://docs.unity3d.com/Manual/index.html.
2. CryEngine V Manual [Електронний ресурс] – Режим доступу: https://docs.cryengine.com/display/CEMANUAL/CRYENGINE+V+Manual.
3. Unreal Engine 4. Engine Features [Електронний ресурс] – Режим доступу: https://docs.unrealengine.com/en-US/Engine/index.html.
4. Бартіш М. Я. Теорія ігор / Бартіш М. Я., Роман Л. Л. – Львів: Видавничий центр ЛНУ, 2005. – 120 с.
5. Battlerite [Електронний ресурс] Режим доступу: https://arena.battlerite.com/.
6. Dota 2 [Електронний ресурс] Режим доступу: http://www.dota2.com/play/.
7. Learning C++ by Creating Games with UE4 by William Sherif. - 2015.
8. Unreal Networking Guide by Zach Metcalf. – 2019.
9. Unreal Engine 4. Blueprints Visual Scripting [Електронний ресурс] – Режим доступу: https://docs.unrealengine.com/en-US/Engine/Blueprints/index.html.
10. Introduction to C++ programming in UE 4 [Електронний ресурс] – Режим доступу: https://docs.unrealengine.com/en-US/Programming/Introduction/index.html.
11. C++ 17 STL Cookbook [Електронний ресурс] – Режим доступу: https://discourse-production.oss-cn-shanghai.aliyuncs.com/original/3X/c/e/0e49f4c82.pdf
12. Unreal Engine 4 Network Compendium [Електронний ресурс] – Режим доступу: http://cedric-neukirchen.net/Downloads/Compendium/UE4_Network_Compendium_by_Cedric_eXi_Neukirchen_BW.pdf
13. Mathematics for 3D Game Programming and Computer Graphics, Third Edition. – 2012

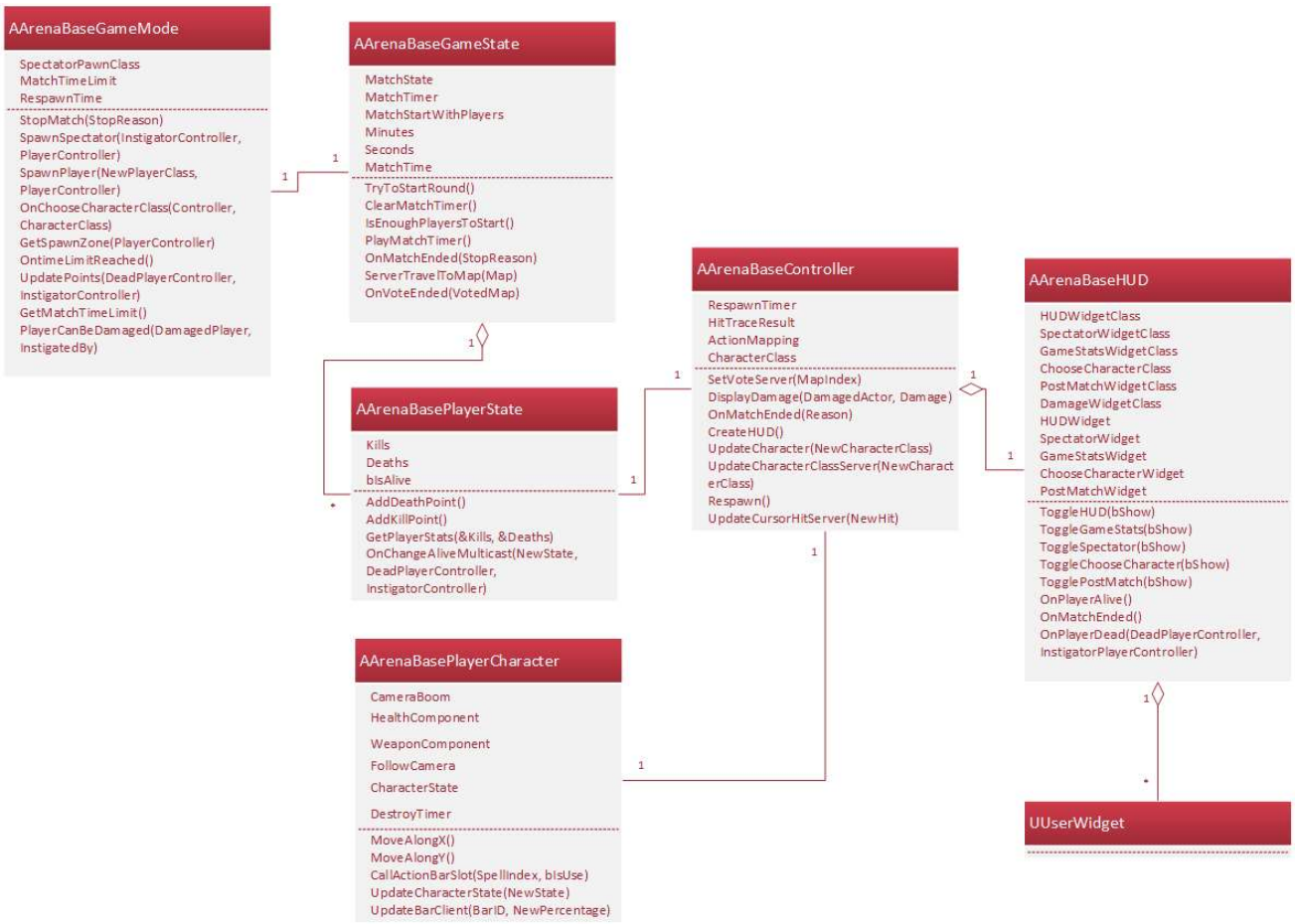
14. Unreal Engine 4. Property specifiers [Електронний ресурс] Режим доступу: https://docs.unrealengine.com/en-US/Programming/UnrealArchitecture/Reference/Properties/Specifiers/index.html.
15. Unreal Engine 4. Function specifiers [Електронний ресурс] Режим доступу: https://docs.unrealengine.com/en-US/Programming/UnrealArchitecture/Reference/Functions/Specifiers/index.html
16. UMG UI Designer Quick Start Guide [Електронний ресурс] Режим доступу: https://docs.unrealengine.com/en-US/Engine/UMG/QuickStart/index.html.
17. Actor Replication in UE 4 [Електронний ресурс] Режим доступу: https://docs.unrealengine.com/en-US/Gameplay/Networking/Actors/index.html.
18. RPC: Protocol Specification [Електронний ресурс] Режим доступу: https://docs.freebsd.org/44doc/psd/26.rpcrfc/paper.pdf
19. Компьютерные игры как феномен современной культуры: опыт междисциплинарного исследования [Електронний ресурс] Режим доступу: http://journals.tsu.ru/uploads/import/1165/files/6-galkin.pdf
20. Basics of AI [Електронний ресурс] Режим доступу: https://www.youtube.com/watch?v=evYE7tfWXUY&t=1s
21. Приклади ігор створених на UE4 [Електронний ресурс] Режим доступу: https://cubiq.ru/dvizhok-unreal-engine/
22. Приклади ігор створених на CryEngine [Електронний ресурс] Режим доступу: https://cubiq.ru/dvizhok-cryengine/
23. Приклади ігор створених на Unity3D [Електронний ресурс] Режим доступу: https://cubiq.ru/dvizhok-cryengine/

Додаток 1

до дипломного проєкту

на тему: «Комп’ютерна гра в жанрі МОВА»

Київ – 2020 року



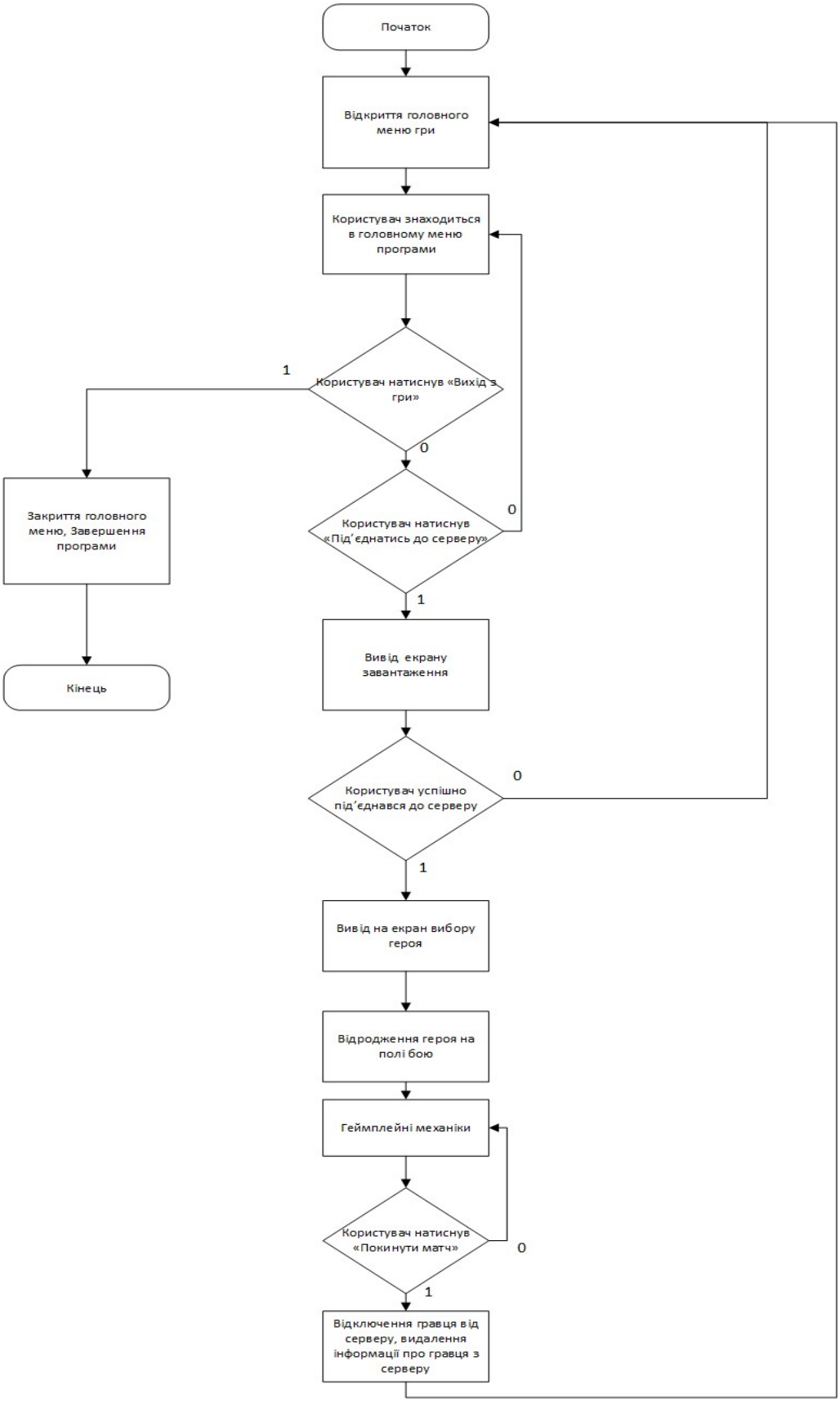
					ІАЛЦ. 467800.004 Д1	
Зм.	Арк.	№ докум.	Підпис	Дата		
Розробив		Бурбіль М.А.			Комп'ютерна гра в жанрі МОВА	Літ. / Аркуш 1 / Аркушів 1
Перевір.					Функціональна схема	
						НТУУ "КПІ ім. Ігоря Сікорського", ФІОТ, ІО-62
Н. контр.		Сімоненко В.П.				
Затверд.						

Додаток 2

до дипломного проєкту

на тему: «Комп’ютерна гра в жанрі МОВА»

Київ – 2020 року



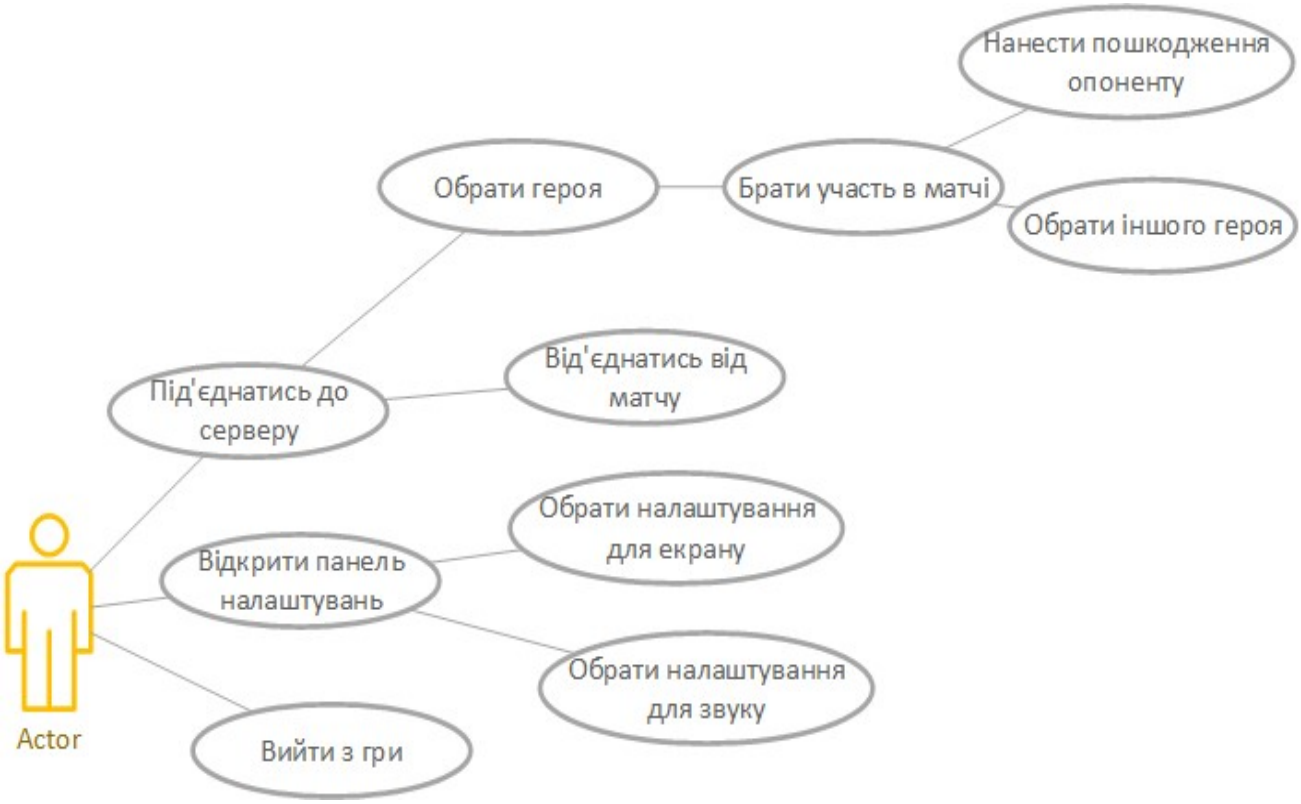
					ІАЛЦ. 467800.005 Д2		
Зм.	*Арк.*	*№ докум.*	*Підпис*	*Дата*			
Розробив		*Бурбіль М.А.*			*Комп'ютерна гра в жанрі МОВА*	*Літ.*	*Аркуш*	*Аркушів*
Перевір.					*Принципова схема*		1	1
						НТУУ "КПІ ім. Ігоря Сікорського", ФІОТ, ІО-62		
Н. контр.		*Сімоненко В.П.*						
Затверд.								

Додаток 3

до дипломного проєкту

на тему: «Комп’ютерна гра в жанрі МОВА»

Київ – 2020 року



					ІАЛЦ. 467800.006 Д3		
Зм.	*Арк.*	*№ докум.*	*Підпис*	*Дата*			
Розробив		*Бурбіль М.А.*			*Комп'ютерна гра в жанрі МОВА*	*Літ.*	*Аркуш*	*Аркушів*
Перевір.					*Структурна схема*		1	1
						НТУУ "КПІ ім. Ігоря Сікорського", ФІОТ, ІО-62		
Н. контр.		*Сімоненко В.П.*						
Затверд.								

Додаток 4

до дипломного проєкту

на тему: «Комп’ютерна гра в жанрі МОВА»

Київ – 2020 року

ТЕКСТ ПРОГРАМИ

```
#include "CoreMinimal.h"
#include "ArenaBaseCharacter.h"
#include "Kismet/KismetMathLibrary.h"
#include "Components/InputComponent.h"
#include "Components/SkeletalMeshComponent.h"
#include "Components/StaticMeshComponent.h"
#include "Components/CapsuleComponent.h"
#include "GameFramework/SpringArmComponent.h"
#include "Camera/CameraComponent.h"
#include "UnrealNetwork.h"
#include "Engine.h"
#include "ArenaCharacterHealthComponent.h"
#include "ArenaWeaponManager.h"
#include "ArenaBaseController.h"
#include "ArenaBaseWeapon.h
AArenaBaseCharacter::AArenaBaseCharacter()
	:
	DestroyTimer(3.f),
	CharacterState(EPlayerState::EPS_Default)
{
	PrimaryActorTick.bCanEverTick = false;
	GetCapsuleComponent()->InitCapsuleSize(40.0f, 88.0f);
	CameraBoom = CreateDefaultSubobject<USpringArmComponent>(TEXT("CameraBoom"));
	CameraBoom->SetupAttachment(RootComponent);
	CameraBoom->bDoCollisionTest = false;
	CameraBoom->bInheritPitch = false;
	CameraBoom->bInheritRoll = false;
	CameraBoom->bInheritYaw = false;
	FollowCamera = CreateDefaultSubobject<UCameraComponent>(TEXT("FollowCamera"));
	FollowCamera->SetupAttachment(CameraBoom, USpringArmComponent::SocketName);
	HealthComponent = CreateDefaultSubobject<UArenaCharacterHealthComponent>(TEXT("Health Component"));
	HealthComponent->SetIsReplicated(true);
	WeaponComponent = CreateDefaultSubobject<UArenaWeaponManager>(TEXT("Weapon Component"));
	WeaponComponent->SetIsReplicated(true);
	LeftWeapon = CreateDefaultSubobject<UStaticMeshComponent>(TEXT("LeftWeapon"));
	LeftWeapon->SetupAttachment(GetMesh());
	LeftWeapon->SetCollisionEnabled(ECollisionEnabled::NoCollision);
	RightWeapon = CreateDefaultSubobject<UStaticMeshComponent>(TEXT("RightWeapon"));
	RightWeapon->SetupAttachment(GetMesh());
	RightWeapon->SetCollisionEnabled(ECollisionEnabled::NoCollision);
	Shield = CreateDefaultSubobject<UStaticMeshComponent>(TEXT("Shield"));
	Shield->SetupAttachment(GetMesh());
	Shield->SetCollisionEnabled(ECollisionEnabled::NoCollision);
	bReplicates = true;
	SetReplicatingMovement(true);
	bNetUseOwnerRelevancy = true;
	bGamepad = false;
}
void AArenaBaseCharacter::PawnClientRestart(){
	Super::PawnClientRestart();
```

```
	AArenaBaseController* MyController = Cast<AArenaBaseController>(Controller);
	if (MyController){
		MyController->CreateHUD();
	}
}
void AArenaBaseCharacter::BeginPlay(){
	Super::BeginPlay();
	if (GetLocalRole() == ROLE_Authority){
		WeaponComponent->CreateSpells();
	}}
void AArenaBaseCharacter::MoveAlongX(float Value){
	if ((Controller) && (Value != 0.0f)){
		AddMovementInput(FVector(1.0f, 0.0f, 0.0f), Value);
	}}
void AArenaBaseCharacter::MoveAlongY(float Value){
	if ((Controller) && (Value != 0.0f)){
		AddMovementInput(FVector(0.0f, 1.0f, 0.0f), Value);
	}}
void AArenaBaseCharacter::CallActionbarSlot(bool bIsUse, int SpellIndex){
	if (SpellIndex < GetWeaponComponent()->CurrentSpells.Num()){
		AArenaBaseWeapon* SelectedSpell = GetWeaponComponent()->CurrentSpells[SpellIndex];
		if (SelectedSpell){
			FName SlotID = SelectedSpell->GetContent().SlotID;
			SetActionbarHighlight.Broadcast(SlotID, bIsUse);
			UseActionbarSpell.Broadcast(SlotID, bIsUse);
		}
		else{
			UE_LOG(LogTemp, Error, TEXT("Spell not found"));
		}}
	else{
		UE_LOG(LogTemp, Error, TEXT("Spell index bigger than CurrentSpells length"));
	}}
void AArenaBaseCharacter::OnRep_CharacterState(){
	switch (CharacterState)  {
	case (EPlayerState::EPS_Default):
		break;
	case (EPlayerState::EPS_Attack):
		break;
	case (EPlayerState::EPS_Dead):
		//UE_LOG(LogTemp, Log, TEXT("ArenaBaseCharacter ONCHARACTER CHANGED: %s"),
*UEnum::GetValueAsString(GetLocalRole()));
		CharacterDead.Broadcast();
		GetCharacterMovement()->DisableMovement();
		GetCapsuleComponent()->SetCollisionEnabled(ECollisionEnabled::NoCollision);
		DisableInput((AArenaBaseController*)Controller);
		GetMesh()->SetCollisionEnabled(ECollisionEnabled::QueryAndPhysics);
		GetMesh()->SetSimulatePhysics(true);
		//On server set life span
		if (GetLocalRole() == ROLE_Authority){
			SetLifeSpan(DestroyTimer);
		}
		break;
```

```
	case (EPlayerState::EPS_Defend):
		break;
	case (EPlayerState::EPS_Dodge):
		break;
	}}
void AArenaBaseCharacter::UpdateBarClient_Implementation(FName BarID, float NewPercentage){
	UpdateBar.Broadcast(BarID, NewPercentage);
}
void AArenaBaseCharacter::UpdateCharacterState(EPlayerState NewState){
	if (GetLocalRole() < ROLE_Authority)	{
		UpdateCharacterStateServer(NewState);
	}
	CharacterState = NewState;
}
void AArenaBaseCharacter::UpdateCharacterStateServer_Implementation(EPlayerState NewState){
	UpdateCharacterState(NewState);
}
bool AArenaBaseCharacter::UpdateCharacterStateServer_Validate(EPlayerState NewState){
	return true;
}
void AArenaBaseCharacter::SetupPlayerInputComponent(UInputComponent* PlayerInputComponent){
	Super::SetupPlayerInputComponent(PlayerInputComponent);
	check(PlayerInputComponent);
	PlayerInputComponent->BindAction("Jump", IE_Pressed, this, &ACharacter::Jump);
	PlayerInputComponent->BindAction("Jump", IE_Released, this, &ACharacter::StopJumping);
	PlayerInputComponent->BindAxis("MoveX", this, &AArenaBaseCharacter::MoveAlongX);
	PlayerInputComponent->BindAxis("MoveY", this, &AArenaBaseCharacter::MoveAlongY);
}

void AArenaBaseCharacter::GetLifetimeReplicatedProps(TArray< FLifetimeProperty > & OutLifetimeProps )
const{
	Super::GetLifetimeReplicatedProps(OutLifetimeProps);
	DOREPLIFETIME(AArenaBaseCharacter, CharacterState);
}
#include "ArenaBasePlayerState.h"
#include "UnrealNetwork.h"
#include "ArenaBaseController.h"
#include "ArenaBaseHUD.h"
AArenaBasePlayerState::AArenaBasePlayerState()
	:
	Kills(0),
	Deaths(0)
{}
void AArenaBasePlayerState::AddDeathPoint(){
	++Deaths;
}
void AArenaBasePlayerState::AddKillPoint(){
	++Kills;
}

void AArenaBasePlayerState::ClientInitialize(AController* C){
	Super::ClientInitialize(C);
```

```
	if (C){
		AArenaBaseController* MyController = Cast<AArenaBaseController>(C);
		if (MyController)		{
			AArenaBaseHUD* MyHUD = Cast<AArenaBaseHUD>(MyController->GetHUD());
			if (MyHUD){
				MyHUD->OnInitialize();
			}}}}
void AArenaBasePlayerState::GetPlayerStats(int& OutKills, int& OutDeaths){
	OutKills = Kills;
	OutDeaths = Deaths;
}
void AArenaBasePlayerState::OnChangeAliveMulticast_Implementation(bool NewState, AArenaBaseController*
DeadPlayerController, AArenaBaseController* InstigatorController){
	bIsAlive = NewState;
	if (bIsAlive)
		OnPlayerAlive.Broadcast();
	else
		OnPlayerDead.Broadcast(DeadPlayerController, InstigatorController);
}
void AArenaBasePlayerState::GetLifetimeReplicatedProps(TArray<FLifetimeProperty>& OutLifetimeProps) const{
	Super::GetLifetimeReplicatedProps(OutLifetimeProps);
	DOREPLIFETIME(AArenaBasePlayerState, bIsAlive);
	DOREPLIFETIME(AArenaBasePlayerState, Kills);
	DOREPLIFETIME(AArenaBasePlayerState, Deaths);
}
#include "ArenaBaseController.h"
#include "Kismet/KismetMathLibrary.h"
#include "Kismet/GameplayStatics.h"
#include "Camera/CameraActor.h"
#include "UnrealNetwork.h"
#include "Engine/World.h"
#include "TimerManager.h"
#include "Blueprint/UserWidget.h"
#include "ArenaGetButtonsLibrary.h"
#include "ArenaBaseCharacter.h"
#include "ArenaBaseGameState.h"
#include "ArenaBaseGameMode.h"
#include "ArenaBaseHUD.h"
#include "ArenaWeaponManager.h"
#include "ArenaBaseWeapon.h"
#include "ArenaVoteComponent.h"
AArenaBaseController::AArenaBaseController(){
	PrimaryActorTick.bCanEverTick = true;
	bShowMouseCursor = true;
	DefaultMouseCursor = EMouseCursor::Crosshairs;
}
void AArenaBaseController::RotateCharacterToMouse(){
	if (AArenaBaseCharacter* ControlledCharacter = Cast<AArenaBaseCharacter>(GetCharacter())){
		FHitResult HitResult;
		if (GetHitResultUnderCursor(ECC_Visibility, true, HitResult)){
			SetControlRotation(FRotator(0,
UKismetMathLibrary::FindLookAtRotation(ControlledCharacter->GetActorLocation(),
```

```
                        HitResult.Location).Yaw, 0));
                HitTraceResult = HitResult;
                UpdateCursorHitServer(HitResult);
        }}}
void AArenaBaseController::Tick(float DeltaTime){
    Super::Tick(DeltaTime);
    if (GetLocalRole() < ROLE_Authority){
        RotateCharacterToMouse();
    }}
void AArenaBaseController::UpdateCharacterClass(TSubclassOf<class AArenaBaseCharacter> NewCharacterClass){
    if (GetLocalRole() < ROLE_Authority){
        UpdateCharacterClassServer(NewCharacterClass);
    }
    CharacterClass = NewCharacterClass;
    AArenaBaseGameState* const GameStateRef = GetWorld() != NULL ? GetWorld()-
>GetGameState<AArenaBaseGameState>() : NULL;
    if (GameStateRef)
        GameStateRef->ChooseCharacter.Broadcast(this, NewCharacterClass);
}
void AArenaBaseController::SetVoteServer_Implementation(int MapIndex){
    AArenaBaseGameState* MyGameState = GetWorld()->GetGameState<AArenaBaseGameState>();
    if (MyGameState){
        MyGameState->VoteComponent->SetVote(MapIndex, PlayerState);
    }}
bool AArenaBaseController::SetVoteServer_Validate(int MapIndex){
    return true;
}
void AArenaBaseController::CreateHUD(){
    AArenaBaseCharacter* MyCharacter = Cast<AArenaBaseCharacter>(GetPawn());
    if (MyCharacter){
        for (int i = 0; i < ActionMapping.Num() - 1; ++i){
            FName Name = FName(*UArenaGetButtonsLibrary::GetMappingButton(GetWorld(),
ActionMapping[i]));
        }
    }
    AArenaBaseHUD* ArenaBaseHUD = Cast<AArenaBaseHUD>(GetHUD());
    if (ArenaBaseHUD){
        ArenaBaseHUD->ToggleHUD(true);
    }}
void AArenaBaseController::DisplayDamage_Implementation(AActor* DamagedActor, float Damage){
    AArenaBaseHUD* MyHUDInstance = Cast<AArenaBaseHUD>(GetHUD());
    if (MyHUDInstance){
        MyHUDInstance->DisplayDamage(DamagedActor, Damage);
    }}
void AArenaBaseController::BeginPlay(){
    Super::BeginPlay();
    if (GetLocalRole() == ROLE_Authority){
        AArenaBaseGameState* MyGameState = GetWorld()->GetGameState<AArenaBaseGameState>();
        if (MyGameState){
            UE_LOG(LogTemp, Log, TEXT("ArenaBaseController BINDING ON MATCH ENDED"));
            MyGameState->MatchEnded.AddDynamic(this, &AArenaBaseController::OnMatchEnded);
        }}}
```

```
void AArenaBaseController::OnMatchEnded(EStopReason Reason){
	AArenaBaseGameState* MyGameState = GetWorld()->GetGameState<AArenaBaseGameState>();
	if (MyGameState){
		MyGameState->MatchEnded.RemoveDynamic(this, &AArenaBaseController::OnMatchEnded);
	}
	GetWorldTimerManager().ClearTimer(RespawnTimer);
}
void AArenaBaseController::UpdateCursorHitServer_Implementation(FHitResult NewHit){
	HitTraceResult = NewHit;
}
bool AArenaBaseController::UpdateCursorHitServer_Validate(FHitResult NewHit){
	return true;
}
void AArenaBaseController::UpdateCharacterClassServer_Implementation(TSubclassOf<class
AArenaBaseCharacter> NewCharacterClass){
	UpdateCharacterClass(NewCharacterClass);
}
bool AArenaBaseController::UpdateCharacterClassServer_Validate(TSubclassOf<class AArenaBaseCharacter>
NewCharacterClass){
	return true;
}
void AArenaBaseController::LaunchRespawnTimer(float RespawnTime){
	GetWorldTimerManager().SetTimer(RespawnTimer,this, &AArenaBaseController::Respawn, RespawnTime,
false);
}
void AArenaBaseController::Respawn(){
	AArenaBaseGameMode* BaseGameMode = Cast<AArenaBaseGameMode>(GetWorld()->GetAuthGameMode());
	if (BaseGameMode){
		BaseGameMode->SpawnPlayer(this, CharacterClass);
		GetWorldTimerManager().ClearTimer(RespawnTimer);
	}
}
void AArenaBaseController::GetLifetimeReplicatedProps(TArray< FLifetimeProperty > & OutLifetimeProps)
const{
	Super::GetLifetimeReplicatedProps(OutLifetimeProps);
	DOREPLIFETIME(AArenaBaseController, CharacterClass);
}
#include "ArcaneArenaEnum.h"
#include "ArenaBaseGameMode.h"
#include "Kismet/GameplayStatics.h"
#include "UnrealNetwork.h"
#include "Engine/World.h"
#include "Engine.h"
#include "ArenaBaseController.h"
#include "ArenaPlayerSpawnZone.h"
#include "ArenaBaseCharacter.h"
#include "ArenaBaseGameState.h"
#include "ArenaBasePlayerState.h"
#include "ArenaSpectatorPawn.h"
#include "Helpers/ArenaCastHelperLibrary.h"
AArenaBaseGameMode::AArenaBaseGameMode()
	:
```

```
	RespawnTime(3.0f),
	MatchTimeLimit(FTimespan(0,0,1,0,0))
{}
bool AArenaBaseGameMode::PlayerCanBeDamaged(AArenaBaseController* DamagedPlayer, AArenaBaseController*
InstigatedBy){
	if (DamagedPlayer && InstigatedBy){
		AArenaBaseCharacter* DamagedCharacter = Cast<AArenaBaseCharacter>(DamagedPlayer->GetPawn());
		if (DamagedCharacter){
			EPlayerState DamagedCharacterState = DamagedCharacter->GetCharacterState();
			return (DamagedCharacterState != EPlayerState::EPS_Dead && DamagedCharacterState !=
EPlayerState::EPS_Defend);
		}}
	return false;
}
void AArenaBaseGameMode::UpdatePoints(AArenaBaseController* DeadPlayerController, AArenaBaseController*
InstigatorController){
	AArenaBasePlayerState* DeadCharacterPlayerState =
UArenaCastHelperLibrary::GetArenaBasePlayerStateByController(DeadPlayerController);
	AArenaBasePlayerState* InstigatorPlayerState =
UArenaCastHelperLibrary::GetArenaBasePlayerStateByController(InstigatorController);
	if (DeadCharacterPlayerState)
		DeadCharacterPlayerState->AddDeathPoint();
	if (InstigatorPlayerState)
		InstigatorPlayerState->AddKillPoint();
}
void AArenaBaseGameMode::StopMatch(EStopReason Reason){
	AArenaBaseGameState* MyGameState = GetWorld()->GetGameState<AArenaBaseGameState>();
	if (MyGameState){
		MyGameState->ClearMatchTimer();
		auto PlayerArray = GameState->PlayerArray;
		for (auto const &PlayerState : PlayerArray){
			auto PlayerController = Cast<AArenaBaseController>(PlayerState->GetOwner());
			if (PlayerController){
				SpawnSpectator(PlayerController, nullptr);
			}
		}
		TArray<AActor*> PlayerCharacters;
		UGameplayStatics::GetAllActorsOfClass(GetWorld(), AArenaBaseCharacter::StaticClass(),
PlayerCharacters);
		for (auto &Character : PlayerCharacters){
			Character->Destroy();
		}
		MyGameState->MatchEnded.Broadcast(Reason);
	}}
void AArenaBaseGameMode::BeginPlay(){
	AArenaBaseGameState* GameStateRef = GetWorld()->GetGameState<AArenaBaseGameState>();
	if (GameStateRef){
		GameStateRef->ChooseCharacter.AddDynamic(this, &AArenaBaseGameMode::OnChooseCharacterClass);
	}
}
void AArenaBaseGameMode::SpawnPlayer(AArenaBaseController* PlayerController, TSubclassOf<class
AArenaBaseCharacter> CharacterClass){
```

```
	if (PlayerController->GetPawn())
		PlayerController->GetPawn()->Destroy();

	AArenaPlayerSpawnZone* SpawnZone = GetSpawnZone(PlayerController);
	if (SpawnZone){
		AArenaBaseCharacter* SpawnedCharacter = SpawnZone->SpawnCharacter(PlayerController,
CharacterClass);
		if (SpawnedCharacter){
			PlayerController->Possess(SpawnedCharacter);
			AArenaBasePlayerState*  BasePlayerState =
Cast<AArenaBasePlayerState>(PlayerController->PlayerState);
			if (BasePlayerState){
				BasePlayerState->bIsAlive = true;
				BasePlayerState->OnChangeAliveMulticast(true, nullptr, nullptr);
			}
			AArenaBaseGameState* MyGameState = GetWorld()->GetGameState<AArenaBaseGameState>();
			if (MyGameState->MatchState == EServerState::MatchPreStart){
				MyGameState->TryToStartRound();
			}}}}
void AArenaBaseGameMode::SpawnSpectator(AArenaBaseController* PlayerController, AArenaBaseController*
InstigatorController){
	if (InstigatorController && InstigatorController != PlayerController){
		AArenaSpectatorPawn* MySpectator = GetWorld()-
>SpawnActorDeferred<AArenaSpectatorPawn>(SpectatorPawnClass,
			PlayerController->GetPawn()->GetActorTransform(), PlayerController, nullptr,
ESpawnActorCollisionHandlingMethod::AlwaysSpawn);
		if (MySpectator){
			MySpectator->SetOwner(PlayerController);
			MySpectator->FollowActor = Cast<AArenaBaseCharacter>(InstigatorController-
>GetPawn());
			MySpectator->AttachedActor = AttachedActor;
			UGameplayStatics::FinishSpawningActor(MySpectator, MySpectator->GetTransform());
			PlayerController->Possess(MySpectator);
		}}
	else{if (AttachedActor){
			AArenaSpectatorPawn* MySpectator = GetWorld()-
>SpawnActorDeferred<AArenaSpectatorPawn>(SpectatorPawnClass,
				AttachedActor->GetActorTransform(), PlayerController, nullptr,
ESpawnActorCollisionHandlingMethod::AlwaysSpawn);
			if (MySpectator){
		MySpectator->SetOwner(PlayerController);
		MySpectator->FollowActor = AttachedActor;
		MySpectator->AttachedActor = AttachedActor;
		UGameplayStatics::FinishSpawningActor(MySpectator, MySpectator->GetTransform());
				PlayerController->Possess(MySpectator);
			}
			else{
				UE_LOG(LogTemp, Error, TEXT("AttachedActor not found"));
			}}}}
void AArenaBaseGameMode::OnCharacterDead(AArenaBaseController* DeadPlayerController, AArenaBaseController*
InstigatorController){
	UpdatePoints(DeadPlayerController, InstigatorController);
```

```
	SpawnSpectator(DeadPlayerController, InstigatorController);
	auto DeadPlayerState =
UArenaCastHelperLibrary::GetArenaBasePlayerStateByController(DeadPlayerController);
	if (DeadPlayerState)
		DeadPlayerState->OnChangeAliveMulticast(false, DeadPlayerController, InstigatorController);
	DeadPlayerController->LaunchRespawnTimer(RespawnTime);
}
void AArenaBaseGameMode::OnTimeLimitReached(){
	StopMatch(EStopReason::ETimeLimit);
}
FTimespan AArenaBaseGameMode::GetMatchTimeLimit() const{
	return MatchTimeLimit;
}
AArenaPlayerSpawnZone * AArenaBaseGameMode::GetSpawnZone(AArenaBaseController * PlayerController){
	TArray<AActor*> SpawnZones;
	UGameplayStatics::GetAllActorsOfClass(GetWorld(), AArenaPlayerSpawnZone::StaticClass(), SpawnZones);
	AArenaPlayerSpawnZone* Zone = SpawnZones.Num() > 0 ?
Cast<AArenaPlayerSpawnZone>(SpawnZones[FMath::RandHelper(SpawnZones.Num())]) : nullptr;
	if (Zone)
		return Zone;
	return nullptr;
}
void AArenaBaseGameMode::OnChooseCharacterClass(AArenaBaseController* Controller, TSubclassOf<class
AArenaBaseCharacter> CharacterClass){
	AArenaBaseGameState* GameStateRef = GetWorld()->GetGameState<AArenaBaseGameState>();
	switch (GameStateRef->MatchState){
	case EServerState::MatchPreStart:
		SpawnPlayer(Controller, CharacterClass);
		break;
	case EServerState::MatchInProgress:
		SpawnPlayer(Controller, CharacterClass);
		break;
	case EServerState::MatchEnd:
		break;
	}}
#include "ArenaBaseGameState.h"
#include "Engine.h"
#include "UnrealNetwork.h"
#include "Engine/DataTable.h"
#include "Engine/World.h"
#include "TimerManager.h"
#include "ArenaBaseGameMode.h"
#include "ArenaBaseHUD.h"
#include "ArenaBaseController.h"
#include "ArenaVoteComponent.h"
AArenaBaseGameState::AArenaBaseGameState(){
	PrimaryActorTick.bCanEverTick = false;
	VoteComponent = CreateDefaultSubobject<UArenaVoteComponent>(TEXT("VoteComponent"));
	bReplicates = true;
}
void AArenaBaseGameState::TryToStartRound(){
	if (IsEnoughPlayersToStart() && MatchState == EServerState::MatchPreStart){
```

```
                MatchState = EServerState::MatchInProgress;
                On_MatchStateChanged();
        }}
void AArenaBaseGameState::ClearMatchTimer(){
        GetWorldTimerManager().ClearTimer(MatchTimer);
}
bool AArenaBaseGameState::IsEnoughPlayersToStart(){
        return (PlayerArray.Num() >= MatchStartWithPlayers);
}
void AArenaBaseGameState::GetMatchTime(int &MinutesOut, int &SecondsOut){
        MinutesOut = Minutes;
        SecondsOut = Seconds;
}
void AArenaBaseGameState::BeginPlay(){
        Super::BeginPlay();
        if (GetLocalRole() == ROLE_Authority){
                MatchEnded.AddDynamic(this, &AArenaBaseGameState::OnMatchEnded);
                AArenaBaseGameMode* MyGameMode = Cast<AArenaBaseGameMode>(GetWorld()->GetAuthGameMode());
                if (MyGameMode)
                        MatchTime = MyGameMode->GetMatchTimeLimit();
                GetWorldTimerManager().SetTimer(MatchTimer, this, &AArenaBaseGameState::PlayMatchTimer,
1.0f, true);
        }
        if (GetLocalRole() < ROLE_Authority){
                AArenaBaseController* MyController =
Cast<AArenaBaseController>(UGameplayStatics::GetPlayerController(GetWorld(), 0));
                if (MyController){
                        AArenaBaseHUD* MyHUD = Cast<AArenaBaseHUD>(MyController->GetHUD());
                        if (MyHUD){
                                MatchEnded.AddDynamic(MyHUD, &AArenaBaseHUD::OnMatchEnded);
                        }}}}
void AArenaBaseGameState::PlayMatchTimer(){
        if (MatchState == EServerState::MatchInProgress){
                MatchTime -= FTimespan(0, 0, 0, 1, 0);
                Minutes = MatchTime.GetMinutes();
                Seconds = MatchTime.GetSeconds();
                if (MatchTime == FTimespan::Zero()){
                        AArenaBaseGameMode* MyGameMode = Cast<AArenaBaseGameMode>(GetWorld()-
>GetAuthGameMode());
                        if (MyGameMode){
                                MyGameMode->OnTimeLimitReached();
                        }}}}
void AArenaBaseGameState::On_MatchStateChanged(){
        switch (MatchState){
                case (EServerState::MatchPreStart):
                        break;
                case (EServerState::MatchInProgress):
                        MatchStarted.Broadcast();
                        break;
                case (EServerState::MatchEnd):
                        break;
        }}
```

```
void AArenaBaseGameState::OnMatchEnded(EStopReason StopReason){
	MatchState = EServerState::MatchEnd;
	MatchEnded.RemoveDynamic(this, &AArenaBaseGameState::OnMatchEnded);
	MatchEndedMulticast(StopReason);
	VoteComponent->VoteStopped.AddDynamic(this, &AArenaBaseGameState::OnVoteEnded);
	VoteComponent->StartVote();
}
void AArenaBaseGameState::OnVoteEnded(FName VotedMap){
	FString ContextString;
	FVoteMapStruct* Row = VoteComponent->MapsTable->FindRow<FVoteMapStruct>(VotedMap, ContextString);
	if (Row){
		ServerTravelToMap(Row->MapName);
	}}
void AArenaBaseGameState::ServerTravelToMap(FName Map){
	FString Command = "ServerTravel " + Map.ToString();
	GetWorld()->Exec(GetWorld(), *Command);
}
void AArenaBaseGameState::MatchEndedMulticast_Implementation(EStopReason StopReason){
	if (GetLocalRole() < ROLE_Authority)
		MatchEnded.Broadcast(StopReason);
}
void AArenaBaseGameState::GetLifetimeReplicatedProps(TArray< FLifetimeProperty > & OutLifetimeProps)
const{
	Super::GetLifetimeReplicatedProps(OutLifetimeProps);
	DOREPLIFETIME(AArenaBaseGameState, Seconds);
	DOREPLIFETIME(AArenaBaseGameState, Minutes);
	DOREPLIFETIME(AArenaBaseGameState, MatchState);
}
#include "ArenaPlayerSpawnZone.h"
#include "Components/BoxComponent.h"
#include "Engine/World.h"
#include "Public/Characters/ArenaBaseCharacter.h"
#include "Public/Controllers/ArenaBaseController.h"
#include "DrawDebugHelpers.h"
AArenaPlayerSpawnZone::AArenaPlayerSpawnZone(){
	PrimaryActorTick.bCanEverTick = false;
	SetReplicates(true);
	SpawnZone = CreateDefaultSubobject<UBoxComponent>(TEXT("SpawnZone"));
	SpawnZone->SetCollisionEnabled(ECollisionEnabled::NoCollision);
	RootComponent = SpawnZone;
}
AArenaBaseCharacter* AArenaPlayerSpawnZone::SpawnCharacter(AArenaBaseController* PlayerController,
TSubclassOf<class AArenaBaseCharacter> CharacterClass){
	FActorSpawnParameters SpawnInfo;
	SpawnInfo.SpawnCollisionHandlingOverride = ESpawnActorCollisionHandlingMethod::AlwaysSpawn;
	SpawnInfo.Owner = PlayerController;
	AArenaBaseCharacter* SpawnedCharacter = GetWorld()->SpawnActor<AArenaBaseCharacter>(CharacterClass,
		GetRandomSpawnPoint() + FVector(0.0f, 0.0f, 100.0f), FRotator::ZeroRotator, SpawnInfo);
	return SpawnedCharacter;
}
FVector AArenaPlayerSpawnZone::GetRandomSpawnPoint(){
	FHitResult HitResult;
```

```
	const FVector WorldLocation = SpawnZone->GetComponentLocation();
	const FVector BoxExtent = SpawnZone->GetScaledBoxExtent();
	const FVector BoxMin = WorldLocation - BoxExtent;
	const FVector BoxMax = WorldLocation + BoxExtent;
	const FVector Start = FMath::RandPointInBox(FBox(BoxMin, BoxMax));
	const FVector End = Start - FVector(0.0f, 0.0f, FMath::Abs(Start.Z * 2.0f));
	if (GetWorld()->LineTraceSingleByChannel(HitResult, Start, End, ECollisionChannel::ECC_Visibility)){
		return HitResult.Location;
	}
	return GetRandomSpawnPoint();
}
#include "ArenaActionbarInterface.h"
#include "Projectiles/ArenaBaseProjectile.h"
#include "Components/CapsuleComponent.h"
#include "Particles/ParticleSystemComponent.h"
#include "GameFramework/ProjectileMovementComponent.h"
AArenaBaseProjectile::AArenaBaseProjectile(){
	PrimaryActorTick.bCanEverTick = true;
	CapsuleCollision = CreateDefaultSubobject<UCapsuleComponent>(TEXT("CapsuleCollision"));
	RootComponent = CapsuleCollision;
	ParticleSystem = CreateDefaultSubobject<UParticleSystemComponent>(TEXT("ParticleSystem"));
	ParticleSystem->SetupAttachment(RootComponent);
	ProjectileMovement =
CreateDefaultSubobject<UProjectileMovementComponent>(TEXT("ProjectileMovement"));
	bReplicates = true;
	SetReplicateMovement(true);
}
#include "ArenaBaseWeapon.h"
#include "Engine.h"
#include "UnrealNetwork.h"
#include "Animation/AnimInstance.h"
#include "ArenaBaseCharacter.h"
#include "ArenaCharacterHealthComponent.h"
#include "ArcaneArenaEnum.h"
AArenaBaseWeapon::AArenaBaseWeapon(){
	PrimaryActorTick.bCanEverTick = false;
	SetReplicates(true);
	SetReplicateMovement(true);
}
bool AArenaBaseWeapon::CanUse(bool bCanBeUsing){
	if (bCanBeUsing && WeaponOwner->GetCharacterState() == EPlayerState::EPS_Default && !bIsCooldown){
		if (IsEnoughEnergy())
			return true;
	}
	return false;
}
bool AArenaBaseWeapon::UseWeapon(bool bIsUse){
	if (GetLocalRole() < ROLE_Authority){
		UseWeaponServer(bIsUse);
	}
	if (CanUse(bIsUse)){
		WeaponOwner->UpdateCharacterState(EPlayerState::EPS_Attack);
```

```
                bIsCooldown = true;
                if (bAutoFire)
                        bIsFireActive = true;
                if (GetLocalRole() < ROLE_Authority){
                        WeaponCooldown.Broadcast();
                }
                else{
                        GetWorldTimerManager().SetTimer(CooldownTimer, this,
&AArenaBaseWeapon::StopCooldown, SlotInfo.Cooldown, false);
                        PlayAnimMontageMulticast();
                }
                return true;
        }
        bIsFireActive = false;
        return false;
}
void AArenaBaseWeapon::UseWeaponServer_Implementation(bool bIsUse){
        UseWeapon(bIsUse);
}
bool AArenaBaseWeapon::UseWeaponServer_Validate(bool bIsUse){
        return true;
}
void AArenaBaseWeapon::StopCooldown(){
        GetWorldTimerManager().ClearTimer(CooldownTimer);
        bIsCooldown = false;
        if (bAutoFire && bIsFireActive)
                UseWeapon(true);
        ClientStopCooldown();
}
bool AArenaBaseWeapon::IsEnoughEnergy(){
        float Energy = WeaponOwner->GetHealthComponent()->GetEnergy();
        if (Energy >= NeededEnergy)
                return true;
        return false;
}
bool AArenaBaseWeapon::IsOtherPlayer(){
        return GetInstigatorController() == nullptr;
}
void AArenaBaseWeapon::PlayAnimMontage(){
        if (WeaponMontage){
                UAnimInstance* Instance = WeaponOwner->GetMesh()->GetAnimInstance();
                if (Instance)
                        Instance->Montage_Play(WeaponMontage, 1.0f);
        }}
void AArenaBaseWeapon::ClientStopCooldown_Implementation(){
        bIsCooldown = false;
        if (bAutoFire && bIsFireActive)
                UseWeapon(true);
}
void AArenaBaseWeapon::Attack_Implementation(FVector SpawnLocation){
}
bool AArenaBaseWeapon::Attack_Validate(FVector SpawnLocation){
```

```
	return true;
}
void AArenaBaseWeapon::PlayAnimMontageMulticast_Implementation(){
	PlayAnimMontage();
}
void AArenaBaseWeapon::Use(bool bUsing){
	if (CanUse(bUsing))
		UseWeapon(true);
	else
		UseWeapon(false);
}
FSlotStruct AArenaBaseWeapon::GetContent(){
	return SlotInfo;
}
void AArenaBaseWeapon::GetLifetimeReplicatedProps(TArray< FLifetimeProperty > & OutLifetimeProps) const{
	Super::GetLifetimeReplicatedProps(OutLifetimeProps);
	DOREPLIFETIME_CONDITION(AArenaBaseWeapon, Damage, COND_OwnerOnly);
	DOREPLIFETIME_CONDITION(AArenaBaseWeapon, NeededEnergy, COND_OwnerOnly);
	DOREPLIFETIME(AArenaBaseWeapon, WeaponOwner);
	DOREPLIFETIME(AArenaBaseWeapon, bIsCooldown);
}
#include "ArenaDodgeWeapon.h"
#include "ArcaneArenaEnum.h"
#include "ArenaBaseCharacter.h"
#include "TimerManager.h"
#include "Engine/World.h"
bool AArenaDodgeWeapon::UseWeapon(bool bIsUse){
	if (GetLocalRole() < ROLE_Authority){
		UseWeaponServer(bIsUse);
	}
	if (CanUse(bIsUse)){
		WeaponOwner->UpdateCharacterState(EPlayerState::EPS_Dodge);
		bIsCooldown = true;
		if (bAutoFire)
			bIsFireActive = true;
		if (GetLocalRole() < ROLE_Authority){
			WeaponCooldown.Broadcast();
		}
		else{
			GetWorldTimerManager().SetTimer(CooldownTimer, this,
&AArenaBaseWeapon::StopCooldown, SlotInfo.Cooldown, false);
			PlayAnimMontageMulticast();
		}
		return true;
	}
	bIsFireActive = false;
	return false;
}
#include "ArenaBaseHUD.h"
#include "HUDWidget.h"
#include "Engine/World.h"
#include "ChooseCharacterWidget.h"
```

```
#include "SpectatorWidget.h"
#include "PostMatchWidget.h"
#include "DamageWidget.h"
#include "GameStatsWidget.h"
#include "ArenaBasePlayerState.h"
#include "ArenaBaseGameState.h"
void AArenaBaseHUD::OnInitialize(){
	AArenaBasePlayerState* MyPlayerState = GetOwningPlayerController()-
>GetPlayerState<AArenaBasePlayerState>();
	if (MyPlayerState){
		MyPlayerState->OnPlayerDead.AddDynamic(this, &AArenaBaseHUD::OnPlayerDead);
		MyPlayerState->OnPlayerAlive.AddDynamic(this, &AArenaBaseHUD::OnPlayerAlive);
	}
	ToggleChooseCharacter(true);
}
void AArenaBaseHUD::DisplayDamage(AActor* DamagedActor, float Damage){
	UDamageWidget* DamageWidget = CreateWidget<UDamageWidget>(GetOwningPlayerController(),
DamageWidgetClass);
	GetOwningPlayerController()->ProjectWorldLocationToScreen(DamagedActor->GetActorLocation(),
DamageWidget->InitialLocation, false);
	DamageWidget->Damage = Damage;
	DamageWidget->AddToViewport();
}
void AArenaBaseHUD::ToggleHUD(bool bShow){
	if (bShow){
		if (HUDWidgetClass){
			if (!HUDWidget){
	HUDWidget = CreateWidget<UHUDWidget>(GetOwningPlayerController(), HUDWidgetClass);
			}
			if (HUDWidget){
				if (!HUDWidget->GetIsVisible()){
					HUDWidget->AddToViewport();
					HUDWidget->SetVisibility(ESlateVisibility::SelfHitTestInvisible);
					GetOwningPlayerController()->bShowMouseCursor = true;
					FInputModeGameAndUI InputMode;

	InputMode.SetLockMouseToViewportBehavior(EMouseLockMode::LockOnCapture);
					InputMode.SetHideCursorDuringCapture(false);
					InputMode.SetWidgetToFocus(HUDWidget->TakeWidget());
					GetOwningPlayerController()->SetInputMode(InputMode);
				}}}}
	else{
		if (HUDWidget){
			HUDWidget->RemoveFromParent();
			HUDWidget->SetVisibility(ESlateVisibility::Hidden);
			FInputModeGameOnly InputMode;
			GetOwningPlayerController()->SetInputMode(InputMode);
			GetOwningPlayerController()->bShowMouseCursor = false;
		}}}
void AArenaBaseHUD::OnPlayerDead(AArenaBaseController* DeadPlayerController, AArenaBaseController*
DeathInstigatorController){
	ToggleChooseCharacter(false);
```

```
	ToggleHUD(false);
	ToggleSpectator(true);
}
void AArenaBaseHUD::OnPlayerAlive(){
	ToggleChooseCharacter(false);
	ToggleSpectator(false);
}
void AArenaBaseHUD::OnMatchEnded(EStopReason Reason){
	ToggleChooseCharacter(false);
	ToggleSpectator(false);
	ToggleHUD(false);
	TogglePostMatch(true);
}
TSubclassOf<UGameStatsWidget> AArenaBaseHUD::GetGameStatsWidgetClass(){
	return GameStatsWidgetClass;
}
#include "CoreMinimal.h"
#include "ArcaneArenaEnum.h"
#include "GameFramework/Character.h"
#include "ArenaBaseCharacter.generated.h"
DECLARE_DYNAMIC_MULTICAST_DELEGATE_TwoParams(FUpdateBarDelegate, FName, BarID, float, NewPercentage);
DECLARE_DYNAMIC_MULTICAST_DELEGATE_TwoParams(FSetActionBarDelegate, FName, BarID, AActor*, Spell);
DECLARE_DYNAMIC_MULTICAST_DELEGATE_TwoParams(FSetHighlightDelegate, FName, BarID, bool, bIsHighlight);
DECLARE_DYNAMIC_MULTICAST_DELEGATE_TwoParams(FUseActionbarSlotDelegate, FName, BarID, bool, bIsUse);
DECLARE_DYNAMIC_MULTICAST_DELEGATE(FOnCharacterDead);
UCLASS()
class ARCANEARENA_API AArenaBaseCharacter : public ACharacter{
	GENERATED_BODY()
UPROPERTY(VisibleAnywhere, BlueprintReadOnly, Category = "Camera", meta = (AllowPrivateAccess = "true"))
	class USpringArmComponent* CameraBoom;
UPROPERTY(VisibleAnywhere, BlueprintReadOnly, Category = "Camera", meta = (AllowPrivateAccess = "true"))
	class UCameraComponent* FollowCamera;
UPROPERTY(VisibleAnywhere, BlueprintReadOnly, Category = "Health", meta = (AllowPrivateAccess = "true"))
	class UArenaCharacterHealthComponent* HealthComponent;
UPROPERTY(VisibleAnywhere, BlueprintReadOnly, Category = "Spells", meta = (AllowPrivateAccess = "true"))
	class UArenaWeaponManager* WeaponComponent;
UPROPERTY(VisibleAnywhere, BlueprintReadOnly, meta = (AllowPrivateAccess = "true"))
		UStaticMeshComponent* RightWeapon;
UPROPERTY(VisibleAnywhere, BlueprintReadOnly, meta = (AllowPrivateAccess = "true"))
		UStaticMeshComponent* LeftWeapon;
	UPROPERTY(VisibleAnywhere, BlueprintReadOnly, meta = (AllowPrivateAccess = "true"))
		UStaticMeshComponent* Shield;
public:
	AArenaBaseCharacter();
	bool bGamepad;
	virtual void PawnClientRestart() override;
protected:
	virtual void BeginPlay() override;
	void MoveAlongX(float Value);
	void MoveAlongY(float Value);
	UFUNCTION(BlueprintCallable, Category = "ArenaBaseCharacter")
		void CallActionbarSlot(bool bIsUse, int SpellIndex);
```

```
	UPROPERTY(EditDefaultsOnly, BlueprintReadWrite, Category = "DestroyOptions")
		float DestroyTimer;
	UPROPERTY(VisibleAnywhere, BlueprintReadWrite, ReplicatedUsing = OnRep_CharacterState, Category =
"States")
		EPlayerState CharacterState;
public:
	UFUNCTION()
	void OnRep_CharacterState();
	virtual void SetupPlayerInputComponent(class UInputComponent* PlayerInputComponent) override;
	UPROPERTY(BlueprintAssignable, BlueprintCallable, Category = "HUD Dispatchers")
		FUpdateBarDelegate UpdateBar;
	UPROPERTY(BlueprintAssignable, BlueprintCallable, Category = "HUD Dispatchers")
		FSetActionBarDelegate SetActionbarContent;
	UPROPERTY(BlueprintAssignable, BlueprintCallable, Category = "HUD Dispatchers")
		FSetHighlightDelegate SetActionbarHighlight;
	UPROPERTY(BlueprintAssignable, BlueprintCallable, Category = "HUD Dispatchers")
		FUseActionbarSlotDelegate UseActionbarSpell;
	UPROPERTY(BlueprintAssignable, BlueprintCallable, Category = "States")
		FOnCharacterDead CharacterDead;
	UFUNCTION(BlueprintCallable, unreliable, Client)
	void UpdateBarClient(FName BarID, float NewPercentage);
	UFUNCTION(BlueprintCallable, Category = "States")
		void UpdateCharacterState(EPlayerState NewState);
	UFUNCTION(Reliable, Server, WithValidation)
		void UpdateCharacterStateServer(EPlayerState NewState);
	FORCEINLINE USpringArmComponent* GetCameraBoom() const { return CameraBoom; }
	FORCEINLINE UCameraComponent* GetFollowCamera() const { return FollowCamera; }
	FORCEINLINE UArenaCharacterHealthComponent* GetHealthComponent() const { return HealthComponent; }
	FORCEINLINE UArenaWeaponManager* GetWeaponComponent() const { return WeaponComponent;  }
	FORCEINLINE EPlayerState GetCharacterState() const { return CharacterState; }
};
#include "CoreMinimal.h"
#include "GameFramework/PlayerState.h"
#include "ArenaBasePlayerState.generated.h"
DECLARE_DYNAMIC_MULTICAST_DELEGATE(FOnPlayerAlive);
DECLARE_DYNAMIC_MULTICAST_DELEGATE_TwoParams(FOnPlayerDead, AArenaBaseController*, DeadPlayerController,
AArenaBaseController*, DeathInstigatorController);
UCLASS()
class ARCANEARENA_API AArenaBasePlayerState : public APlayerState{
	GENERATED_BODY()
public:
    AArenaBasePlayerState();
    UPROPERTY(BlueprintAssignable, BlueprintCallable, Category = "ArenaPlayerState")
        FOnPlayerAlive OnPlayerAlive;
    UPROPERTY(BlueprintAssignable, BlueprintCallable, Category = "ArenaPlayerState")
        FOnPlayerDead OnPlayerDead;
    UPROPERTY(VisibleAnywhere, BlueprintReadOnly, Replicated, Category = "ArenaPlayerState", meta =
(ToolTip = "Changes when character dies or respawns"))
        bool bIsAlive;
    void AddDeathPoint();
    void AddKillPoint();
    virtual void ClientInitialize(AController* C) override;
```

```
    UFUNCTION(NetMulticast, Reliable)
        void OnChangeAliveMulticast(bool NewState, AArenaBaseController* DeadPlayerController,
AArenaBaseController* InstigatorController);
    UFUNCTION(BlueprintCallable, BlueprintPure, Category = "ArenaPlayerState")
        void GetPlayerStats(int& OutKills, int& OutDeaths);
private:
    UPROPERTY(Replicated)
    int Kills;
    UPROPERTY(Replicated)
    int Deaths;
};
#include "CoreMinimal.h"
#include "GameFramework/PlayerController.h"
#include "ArenaBaseController.generated.h"
UCLASS()
class ARCANEARENA_API AArenaBaseController : public APlayerController{
      GENERATED_BODY()
public:
      AArenaBaseController();
      UPROPERTY(BlueprintReadOnly, Category = "ArenaBaseController")
              FHitResult HitTraceResult;
      FTimerHandle RespawnTimer;
      UPROPERTY(EditDefaultsOnly, BlueprintReadOnly, Category = "ArenaBaseController.ActionsMapping")
              TArray<FName> ActionMapping;
      virtual void Tick(float DeltaTime) override;
      void LaunchRespawnTimer(float RespawnTime);
      void RotateCharacterToMouse();
      UFUNCTION()
              void Respawn();
      UFUNCTION(BlueprintCallable, Category = "Character Options")
              void UpdateCharacterClass(TSubclassOf<AArenaBaseCharacter> NewCharacterClass);
      UFUNCTION(BlueprintCallable, Reliable, Server, WithValidation)
              void UpdateCharacterClassServer(TSubclassOf<AArenaBaseCharacter> NewCharacterClass);
      UFUNCTION(BlueprintCallable, Reliable, Server, WithValidation)
              void SetVoteServer(int MapIndex);
      UFUNCTION(BlueprintCallable, Category = "ArenaBaseController.HUD")
              void CreateHUD();
      UFUNCTION(Client, Unreliable)
              void DisplayDamage(AActor* DamagedActor, float Damage);
      FORCEINLINE TSubclassOf<AArenaBaseCharacter> GetCharacterClass() const { return CharacterClass; }
protected:
      UPROPERTY(VisibleAnywhere, BlueprintReadOnly, Replicated)
              TSubclassOf<AArenaBaseCharacter> CharacterClass;
      virtual void BeginPlay() override;
      UFUNCTION()
              void OnMatchEnded(EStopReason Reason);
private:
      UFUNCTION(Server, Reliable, WithValidation)
              void UpdateCursorHitServer(FHitResult NewHit);
};
#include "CoreMinimal.h"
#include "GameFramework/GameModeBase.h"
```

```
#include "ArcaneArenaEnum.h"
#include "Misc/Timespan.h"
#include "ArenaBaseGameMode.generated.h"
UCLASS()
class ARCANEARENA_API AArenaBaseGameMode : public AGameModeBase{
	GENERATED_BODY()
public:
	AArenaBaseGameMode();
UPROPERTY(VisibleAnywhere, BlueprintReadWrite, Category = "ArenaGameMode")
		AActor* AttachedActor;
virtual bool PlayerCanBeDamaged (AArenaBaseController* DamagedPlayer, AArenaBaseController* InstigatedBy);
void UpdatePoints(AArenaBaseController* DeadPlayerController, AArenaBaseController* InstigatorController);
void SpawnPlayer(AArenaBaseController* PlayerController, TSubclassOf<AArenaBaseCharacter> CharacterClass);
void OnCharacterDead(AArenaBaseController* DeadPlayerController, AArenaBaseController*
InstigatorController);
	void OnTimeLimitReached();
	FTimespan GetMatchTimeLimit() const;
protected:
	UPROPERTY(EditAnywhere, Category = "ArenaGameMode")
		TSubclassOf<AArenaSpectatorPawn> SpectatorPawnClass;
	virtual void BeginPlay() override;
	void SpawnSpectator(AArenaBaseController* PlayerController, AArenaBaseController*
InstigatorController);
	AArenaPlayerSpawnZone* GetSpawnZone(AArenaBaseController* PlayerController);
	UFUNCTION()
void OnChooseCharacterClass(AArenaBaseController* Controller, TSubclassOf<AArenaBaseCharacter>
CharacterClass);
private:
	UPROPERTY(EditDefaultsOnly, Category = "ArenaGameMode.MatchSettings")
		float RespawnTime;
	UPROPERTY(EditDefaultsOnly, Category = "ArenaGameMode.MatchSettings")
		FTimespan MatchTimeLimit;
	void StopMatch(EStopReason Reason);
};
#include "CoreMinimal.h"
#include "GameFramework/GameStateBase.h"
#include "ArcaneArenaEnum.h"
#include "Misc/Timespan.h"
#include "ArenaBaseGameState.generated.h"
DECLARE_DYNAMIC_MULTICAST_DELEGATE(FOnMatchStarted);
DECLARE_DYNAMIC_MULTICAST_DELEGATE_OneParam(FOnMatchEnded, EStopReason, StopReason);
DECLARE_DYNAMIC_MULTICAST_DELEGATE_TwoParams(FOnChooseCharacter, AArenaBaseController*, Controller,
TSubclassOf<AArenaBaseCharacter>, CharacterClass);
UCLASS()
class ARCANEARENA_API AArenaBaseGameState : public AGameStateBase{
	GENERATED_BODY()
public:
	AArenaBaseGameState();
	UPROPERTY(VisibleAnywhere, BlueprintReadWrite, ReplicatedUsing = On_MatchStateChanged, Category =
"ArenaBaseGameState.MatchSettings")
		UArenaVoteComponent* VoteComponent;
	UPROPERTY(BlueprintAssignable, BlueprintCallable, Category = "ArenaBaseGameState.MatchStates")
```

```
		FOnMatchStarted MatchStarted;
	UPROPERTY(BlueprintAssignable, BlueprintCallable, Category = "ArenaBaseGameState.MatchStates")
		FOnMatchEnded MatchEnded;
	UPROPERTY(BlueprintAssignable)
		FOnChooseCharacter ChooseCharacter;
	UPROPERTY(VisibleAnywhere, BlueprintReadWrite, ReplicatedUsing = On_MatchStateChanged, Category =
"ArenaBaseGameState.MatchSettings")
		EServerState MatchState;
	void TryToStartRound();
	void ClearMatchTimer();
	UFUNCTION(BlueprintCallable, BlueprintPure, Category = "ArenaBaseGameState")
		bool IsEnoughPlayersToStart();
	UFUNCTION(BlueprintCallable, BlueprintPure, Category = "ArenaBaseGameState.MatchTime")
		void GetMatchTime(int &MinutesOut, int &SecondsOut);
protected:
	FTimerHandle MatchTimer;
	UPROPERTY(EditDefaultsOnly, BlueprintReadWrite, Category = "ArenaBaseGameState.MatchSettings")
	int MatchStartWithPlayers;
	virtual void BeginPlay() override;
	UFUNCTION()
		void PlayMatchTimer();
	UFUNCTION()
		void On_MatchStateChanged();
	UFUNCTION()
		void OnMatchEnded(EStopReason StopReason);
	UFUNCTION()
		void OnVoteEnded(FName VotedMap);
	UFUNCTION(NetMulticast, Reliable)
		void MatchEndedMulticast(EStopReason StopReason);
	void ServerTravelToMap(FName Map);
private:
	UPROPERTY(VisibleAnywhere, Replicated, Category = "ArenaBaseGameState.MatchTime")
		int Minutes;
	UPROPERTY(VisibleAnywhere, Replicated, Category = "ArenaBaseGameState.MatchTime")
		int Seconds;
	UPROPERTY(VisibleAnywhere, Category = "ArenaBaseGameState.MatchTime")
		FTimespan MatchTime;
};
#include "CoreMinimal.h"
#include "ArenaActionbarInterface.h"
#include "GameFramework/Actor.h"
#include "ArenaBaseWeapon.generated.h"
DECLARE_DYNAMIC_MULTICAST_DELEGATE(FOnWeaponCooldown);
UCLASS()
class ARCANEARENA_API AArenaBaseWeapon : public AActor, public IArenaActionbarInterface{
	GENERATED_BODY()
public:
	AArenaBaseWeapon();
	UPROPERTY(BlueprintReadOnly, Replicated, Category = "ArenaBaseWeapon")
		AArenaBaseCharacter* WeaponOwner;
	UPROPERTY(BlueprintAssignable, Category = "States")
		FOnWeaponCooldown WeaponCooldown;
```

```
	UPROPERTY(EditDefaultsOnly, BlueprintReadOnly, Category = "ArenaBaseWeapon|Settings|WeaponInfo")
		FSlotStruct SlotInfo;
	void StopCooldown();
	bool CanUse(bool bCanBeUsing);
	virtual void Use(bool bUsing) override;
	virtual FSlotStruct GetContent() override;
	virtual bool UseWeapon(bool bIsUse);
	UFUNCTION(Server, Reliable, WithValidation)
		void UseWeaponServer(bool bIsUse);
	UFUNCTION(BlueprintCallable, Server, Reliable, WithValidation, Category = "ArenaBaseWeapon")
		virtual void Attack(FVector SpawnLocation);
protected:
	FTimerHandle CooldownTimer;
	UPROPERTY(VisibleAnywhere, BlueprintReadOnly, Replicated, Category = "States")
		bool bIsCooldown;
	UPROPERTY(EditAnywhere, BlueprintReadWrite, Category = "States")
		bool bAutoFire;
	UPROPERTY(VisibleAnywhere, BlueprintReadOnly, Category = "States")
		bool bIsFireActive;
	UPROPERTY(VisibleAnywhere, BlueprintReadOnly, Category = "States")
		bool bIsStopSpawn;
	UPROPERTY(EditAnywhere, BlueprintReadWrite, Replicated, Category =
"ArenaBaseWeapon|Settings|WeaponInfo")
		float Damage;
	UPROPERTY(EditAnywhere, BlueprintReadWrite, Replicated, Category =
"ArenaBaseWeapon|Settings|WeaponInfo")
		float NeededEnergy;
	UPROPERTY(EditAnywhere, BlueprintReadWrite, Category = "ArenaBaseWeapon|Settings|WeaponInfo")
		UAnimMontage* WeaponMontage;
	bool IsEnoughEnergy();
	bool IsOtherPlayer();
	void PlayAnimMontage();
	UFUNCTION(NetMulticast, Reliable, Category = "ArenaBaseWeapon")
		void PlayAnimMontageMulticast();
	UFUNCTION(Client, Reliable, Category = "ArenaBaseWeapon")
		void ClientStopCooldown();
};
```